FOREWORD

JEAN-YVES MARION[1] AND THOMAS SCHWENTICK[2]

[1] Nancy University,LORIA
E-mail address: Jean-Yves.Marion@loria.fr

[2] TU Dortmundt
E-mail address: Thomas.Schwentick@udo.edu

The Symposium on Theoretical Aspects of Computer Science (STACS) is held alternately in France and in Germany. The conference of March 4-6, 2010, held in Nancy, is the 27th in this series. Previous meetings took place in Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), Montpellier (2004), Stuttgart (2005), Marseille (2006), Aachen (2007), Bordeaux (2008), and Freiburg (2009). The interest in STACS has remained at a high level over the past years. The STACS 2010 call for papers led to over 238 submissions from 40 countries. Each paper was assigned to three program committee members. The committee selected 54 papers during a two- week electronic meeting held in November. As co-chairs of the program committee, we would like to sincerely thank its members and the many external referees for their valuable work. In particular, there were intense and interesting discussions. The overall very high quality of the submissions made the selection a difficult task. We would like to express our thanks to the three invited speakers, Mikołaj Bojańczyk, Rolf Niedermeier, and Jacques Stern. Special thanks go to Andrei Voronkov for his EasyChair software (www.easychair.org). Moreover, we would like to warmly thank Wadie Guizani for preparing the conference proceedings and continuous help throughout the conference organization. For the third time, this year's STACS proceedings are published in electronic form. A printed version was also available at the conference, with ISBN. The electronic proceedings are available through several portals, and in particular through HAL and LIPIcs series . The proceedings of the Symposium, which are published electronically in the LIPIcs (Leibniz International Proceedings in Informatics) series, are available through Dagstuhl's website. The LIPIcs series provides an ISBN for the proceedings volume and manages the indexing issues. HAL is an electronic repository managed by several French research agencies. Both, HAL and the LIPIcs series, guarantee perennial, free and easy electronic access, while the authors will retain the rights over their work. The rights on the articles in the proceedings are kept with the authors and the papers are available freely, under a Creative Commons license (see www.stacs- conf.org/faq.html for more details).

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE
27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2439

STACS 2010 received funds from Nancy-University (UHP, Nancy 2 and INPL), from Région Lorraine, from CUGN, from GIS 3SG, from GDR IM and from Mairie de Nancy. We thank them for their support!

February 2010 Jean-Yves Marion and Thomas Schwentick

Conference Organisation

STACS 2010 was organized by INRIA Nancy-Grand-Est at LORIA, Nancy University.

Members of the program committee

Markus Bläser	Saarland University
Harry Buhrman	CWI, Amsterdam University
Thomas Colcombet	CNRS, Paris 7 University
Anuj Dawar	University of Cambridge
Arnaud Durand	Paris 7 University
Sándor Fekete	Braunschweig University of Technology
Ralf Klasing	CNRS, Bordeaux University
Christian Knauer	Freie Universität Berlin
Piotr Krysta	University of Liverpool
Sylvain Lombardy	Marne la vallée University
P. Madhusudan	University of Illinois
Jean-Yves Marion	Nancy University (co-chair)
Pierre McKenzie	University of Montréal
Rasmus Pagh	IT University of Copenhagen
Boaz Patt-Shamir	Tel Aviv University
Christophe Paul	CNRS, Montpellier University
Georg Schnitger	Frankfurt University
Thomas Schwentick	TU Dortmund University (co-chair)
Helmut Seidl	TU Munich
Jiří Sgall	Charles University
Sebastiano Vigna	Universitá degli Studi di Milano
Paul Vitanyi	CWI, Amsterdam

Members of the organizing committee

Nicolas Alcaraz
Anne-Lise Charbonnier
Jean-Yves Marion
Wadie Guizani

External Reviewers

Ittai Abraham
Eyal Ackerman
Manindra Agrawal
Stefano Aguzzoli
Cyril Allauzen
Eric Allender
Noga Alon
Alon Altman
Andris Ambainis
Amihood Amir
Eric Angel
Esther Arkin
Diego Arroyuelo
Eugene Asarin
Albert Atserias
Nathalie Aubrun
Laszlo Babai
Patrick Baillot
Joergen Bang-Jensen
Vince Barany
Jérémy Barbay
Georgios Barmpalias
Clark Barrett
David Mix Barrington
Luca Becchetti
Wolfgang Bein
Djamal Belazzougui

Anne Benoit
Piotr Berman
alberto bertoni
Philippe Besnard
Stéphane Bessy
Laurent Bienvenu
Philip Bille
Davide Bilò
Henrik Björklund
Guillaume Blin
Hans Bodlaender
Hans-Joachim Boeckenhauer
Guillaume Bonfante
Vincenzo Bonifaci
Yacine Boufkhad
Laurent Boyer
Zvika Brakerski
Felix Brandt
Jop Briet
Kevin Buchin
Maike Buchin
Andrei Bulatov
Jaroslaw Byrka
Marie-Pierre Béal
Sergio Cabello
Michaël Cadilhac
Arnaud Carayol
Olivier Carton
Giovanni Cavallanti
Rohit Chadha
Amit Chakrabarti
Sourav Chakraborty
Jérémie Chalopin
Jean-Marc Champarnaud
Pierre Charbit
Krishnendu Chatterjee
Arkadev Chattopadhyay
Chandra Chekuri
Ho-Lin Chen
James Cheney
Victor Chepoi
Alessandra Cherubini
Flavio Chierichetti
Giorgos Christodoulou
Marek Chrobak
Richard Cleve
Éric Colin de Verdière
Colin Cooper
Graham Cormode
Veronique Cortier
Bruno Courcelle
Nadia Creignou
Maxime Crochemore
Jurek Czyzowicz
Flavio D'Alessandro
Jean Daligault
Victor Dalmau
Shantanu Das
Samir Datta
Fabien de Montgolfier
Michel de Rougemont
Søren Debois
Holger Dell
Camil Demetrescu
Britta Denner-Broser
Bilel Derbel
Jonathan Derryberry
Josee Desharnais
Luc Devroye
Claudia Dieckmann
Scott Diehl
Martin Dietzfelbinger
Frank Drewes
Andy Drucker
Philippe Duchon
Adrian Dumitrescu
Jérôme Durand-Lose
David Duris
Stephane Durocher
Ivo Düntsch
Christian Eisentraut
Yuval Emek
Matthias Englert
David Eppstein
Leah Epstein
Thomas Erlebach
Omid Etesami
Kousha Etessami
Guy Even
Rolf Fagerberg
Michael Fellows
Stefan Felsner
Jiri Fiala
Amos Fiat
Bernd Finkbeiner
Irene Finocchi
Felix Fischer
Jörg Flum
Fedor Fomin
Lance Fortnow
Hervé Fournier
Mahmoud Fouz
Pierre Fraigniaud
Gianni Franceschini
Stefan Funke
Nicola Galesi
Philippe Gambette
David Garcia Soriano
Leszek Gasieniec
Serge Gaspers
Serge Gaspers
Bruno Gaujal
Cyril Gavoille
Wouter Gelade
Dirk H.P. Gerrits
Panos Giannopoulos
Richard Gibbens
Hugo Gimbert
Emeric gioan
Christian Glasser
Leslie Ann Goldberg
Paul Goldberg
Rodolfo Gomez
Robert Grabowski
Fabrizio Grandoni
Frederic Green
Serge Grigorieff
Erich Grädel
Joachim Gudmundsson
Sylvain Guillemot
Pierre Guillon
Yuri Gurevich
Venkatesan Guruswami
Peter Habermehl
Gena Hahn
MohammadTaghi Hajiaghayi
Sean Hallgren
Michal Hanckowiak
Sariel Har-Peled
Moritz Hardt
Tero Harju

Matthias Hein
Raymond Hemmecke
Miki Hermann
Danny Hermelin
John Hitchcock
Martin Hoefer
Christian Hoffmann
Frank Hoffmann
Thomas Holenstein
Markus Holzer
Peter Hoyer
Mathieu Hoyrup
Jing Huang
Paul Hunter
Thore Husfeldt
Marcus Hutter
Nicole Immorlica
Shunsuke Inenaga
Riko Jacob
Andreas Jakoby
Alain Jean-Marie
Mark Jerrum
Gwenaël Joret
Stasys Jukna
Valentine Kabanets
Lukasz Kaiser
Tom Kamphans
Mamadou Kanté
Mamadou Moustapha Kanté
Jarkko Kari
Veikko Keranen
Sanjeev Khanna
Stefan Kiefer
Alex Kipnis
Adam Klivans
Johannes Koebler
Natallia Kokash
Petr Kolman
Jochen Konemann
Miroslaw Korzeniowski
Adrian Kosowski
Michal Koucky
Michal Koucky
Matjaz Kovse
Máté Kovács
Jan Krajicek
Daniel Kral
Jan Kratochvil
Dieter Kratsch
Stefan Kratsch
Robi Krauthgamer
Steve Kremer
Klaus Kriegel
Danny Krizanc
Alexander Kroeller
Andrei Krokhin
Gregory Kucherov
Denis Kuperberg
Tomi Kärki
Juha Kärkkäinen
Ekkehard Köhler
Salvatore La Torre
Arnaud Labourel
Gad Landau
Jérôme Lang
Sophie Laplante
Benoit Larose
Silvio Lattanzi
Lap Chi Lau
Soeren Laue
Thierry Lecroq
Troy Lee
Arnaud Lefebvre
Aurelien Lemay
François Lemieux
Benjamin Leveque
Asaf Levin
Mathieu Liedloff
Andrzej Lingas
Tadeusz Litak
Christof Loeding
Daniel Lokshtanov
Tzvi Lotker
zvi lotker
Laurent Lyaudet
Florent Madelaine
Frederic Magniez
Meena Mahajan
Anil Maheshwari
Johann Makowsky
Guillaume Malod
Sebastian Maneth
Yishay Mansour
Roberto Mantaci
Bodo Manthey
Martin Mares
Maurice Margenstern
Euripides Markou
Wim Martens
Barnaby Martin
Kaczmarek Matthieu
Frédéric Mazoit
Damiano Mazza
Carlo Mereghetti
Julian Mestre
Peter Bro Miltersen
Vahab Mirrokni
Joseph Mitchell
Tobias Moemke
Stefan Monnier
Ashley Montanaro
Thierry Monteil
Pat Morin
Hannes Moser
Larry Moss
Luca Motto Ros
Marie-Laure Mugnier
Wolfgang Mulzer
Andrzej Murawski
Filip Murlak
Viswanath Nagarajan
Rouven Naujoks
Jesper Nederlof
Yakov Nekrich
Ilan Newman
Cyril Nicaud
Shuxin Nie
Evdokia Nikolova
Aviv Nisgav
Jean Néraud
Marcel Ochel
Sergei Odintsov
Nicolas Ollinger
Alessio Orlandi
Friedrich Otto
Martin Otto
Sang-il Oum
Linda Pagli
Beatrice Palano
Ondrej Pangrac
Rina Panigrahy

Gennaro Parlato
Arno Pauly
Anthony Perez
Martin Pergel
Sylvain Perifel
Rafael Peñaloza
Giovanni Pighizzini
Nir Piterman
David Podgorolec
Vladimir Podolskii
Natacha Portier
Sylvia Pott
Victor Poupet
Christophe Prieur
Ariel Procaccia
Guido Proietti
Pavel Pudlak
Arnaud Pêcher
Tomasz Radzik
Anup Rao
Dror Rawitz
Saurabh Ray
Christian Reitwießner
Eric Remila
Mark Reynolds
Ahmed Rezine
Eric Rivals
Romeo Rizzi
Julien Robert
Peter Rossmanith
Jacques Sakarovitch
Mohammad Salavatipour
Kai Salomaa
Louis Salvail
Marko Samer
Nicola Santoro
Srinivasa Rao Satti
Ignasi Sau
Thomas Sauerwald
Saket Saurabh
Rahul Savani
Petr Savicky
Gabriel Scalosub
Guido Schaefer
Marc Scherfenberg
Lena Schlipf
Stefan Schmid
Christiane Schmidt
Jens M. Schmidt
Henning Schnoor
Warren Schudy
Nils Schweer
Pascal Schweitzer
Daria Schymura
Bernhard Seeger
Raimund Seidel
Pranab Sen
Siddhartha Sen
Olivier Serre
Rocco Servedio
Anil Seth
Alexander Sherstov
Amir Shpilka
Rene Sitters
Alexander Skopalik
Nataliya Skrypnyuk
Michiel Smid
Michiel Smid
Jack Snoeyink
Christian Sohler
Jeremy Sproston
Fabian Stehn
Clifford Stein
Sebastian Stiller
Yann Strozecki
Subhash Suri
Chaitanya Swamy
Till Tantau
Alain Tapp
Anusch Taraz
Nina Sofia Taslaman
Monique Teillaud
Pascal Tesson
Guillaume Theyssier
Dimitrios Thilikos
Wolfgang Thomas
Mikkel Thorup
Christopher Thraves
Ramki Thurimella
Alwen Tiu
Hans Raj Tiwary
Sebastien Tixeuil
Ioan Todinca
Craig Tovey
A.N. Trahtman
Luca Trevisan
Nicolas Trotignon
Falk Unger
Walter Unger
Sarvagya Upadhyay
Wim van Dam
Peter van Emde Boas
Dieter van Melkebeek
Rob van Stee
Anke van Zuylen
Yann Vaxès
Rossano Venturini
Kolia Vereshchagin
Stéphane Vialette
Ivan Visconti
Smitha Vishveshwara
Mahesh Viswanathan
Heribert Vollmer
Uli Wagner
Igor Walukiewicz
Rolf Wanka
Egon Wanke
Mark Daniel Ward
Osamu Watanabe
John Watrous
Roger Wattenhofer
Tzu-chieh Wei
Daniel Werner
Ryan Williams
Erik Winfree
Gerhard Woeginger
Philipp Woelfel
Dominik Wojtczak
Paul Wollan
James Worrell
Sai Wu
Andrew C.-C. Yao
Sergey Yekhanin
Ke Yi
Jean-Baptiste Yunès
Raphael Yuster
Konrad Zdanowski
Mariano Zelke
Akka Zemmari
Uri Zwick.

TABLE OF CONTENTS

Invited Talks

Contributed Papers

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2505

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 11-16
www.stacs-conf.org

BEYOND ω-REGULAR LANGUAGES

MIKOŁAJ BOJAŃCZYK

University of Warsaw
E-mail address: bojan@mimuw.edu.pl
URL: www.mimuw.edu.pl/∼bojan

Abstract. The paper presents some automata and logics on ω-words, which capture all ω-regular languages, and yet still have good closure and decidability properties.

The notion of ω-regular language is well established in the theory of automata. The class of ω-regular languages carries over to ω-words many of the good properties of regular languages of finite words. It can be described using automata, namely by nondeterministic Büchi automata, or the equivalent deterministic Muller automata. It can be described using a form of regular expressions, namely by ω-regular expressions. It can be described using logic, namely by monadic second-order logic, or the equivalent weak monadic-second order logic.

This paper is about some recent work [1, 3, 2, 4], which argues that there are other robust classes of languages for ω-words. The following languages serve as guiding examples.

$$L_B = \{a^{n_1}ba^{n_2}b\cdots : \limsup n_i < \infty\} \qquad L_S = \{a^{n_1}ba^{n_2}b\cdots : \liminf n_i = \infty\}$$

Neither of these languages is ω-regular in the accepted sense. One explanation is that L_S contains no ultimately periodic word, as does the complement of L_B. Another explanation is that an automaton recognizing either of these languages would need an infinite amount of memory, to compare the numbers $n_1, n_2, \ldots$

Both of these explanations can be disputed.

Concerning the first explanation: why should ultimately periodic words be so important? Clearly there are other finite ways of representing infinite words. A nonempty Büchi automaton will necessarily accept an ultimately periodic word, and hence their importance in the theory of ω-regular languages. But is this notion canonic? Or is it just an artefact of the syntax we use?

Concerning the second explanation: what does "infinite memory" mean? After all, one could also argue that the ω-regular language $(a^*b)^\omega$ needs infinite memory, to count the b's that need to appear infinitely often. In at least one formalization of "memory", the languages L_B and L_S do not need infinite memory. The formalization uses a Myhill-Nerode style equivalence. For a language $L \subseteq A^\omega$, call two finite words L-equivalent if they can be

Key words and phrases: automata, monadic second-order logic.
Author supported by ERC Starting Grant "Sosna".

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2440

swapped a finite or infinite number of times without L noticing. Formally, words $w, v \in A^*$ are called L-equivalent if both conditions below hold.

$$\begin{aligned} u_1 w u_2 \in L &\iff u_1 v u_2 \in L && \text{for } u_1 \in A^*, u_2 \in A^\omega \\ u_1 w u_2 w u_3 w \cdots \in L &\iff u_1 v u_2 v u_3 v \cdots \in L && \text{for } u_1, u_2, \ldots \in A^*. \end{aligned}$$

One can show that L_B-equivalence has three equivalence classes, and L_S-equivalence has four equivalence classes. Therefore, at least in this Myhill-Nerode sense, the languages L_B and L_S do not need infinite memory.

The rest of this paper presents some language classes which capture L_B and L_S, and which have at least some of the robustness properties one would expect from regular languages. We begin with a logic.

MSO with the unbounding quantifier. Monadic second-order logic (MSO) captures exactly the ω-regular languages. To define the languages L_B and L_S, some new feature is needed. Consider a new quantifier $\mathsf{U}X\ \varphi(X)$, introduced in [1], which says that formula $\varphi(X)$ is satisfied by arbitrarily large finite sets X, i.e.

$$\mathsf{U}X\ \varphi(X) = \bigwedge_{n \in \mathbb{N}} \exists X\ \big(\ \varphi(X) \quad \wedge \quad n \le |X| < \infty\ \big).$$

As usual with quantifiers, the formula $\varphi(X)$ might have other free variables than X. We write MSO+U for the extension of MSO where this quantifier is allowed. It is difficult to say if U is an existential or universal quantifier, since its definition involves an infinite conjunction of existential formulas.

Let us see some examples of formulas of MSO+U. Consider a formula $block(X)$ which says that X contains all positions between two consecutive b's. To define the language L_B in the logic MSO+U, we need to say that: i) there are infinitely many b's and ii) the size of blocks is not unbounded. This is done by the following formula.

$$\forall x \exists y (x \le y \wedge b(y)) \quad \wedge \quad \neg \mathsf{U}X\ block(X).$$

For the language L_S, we need a more sophisticated formula. It is easier to write a formula for the complement of L_S. The formula says that there exists a set Z, which contains infinitely many blocks, as stated by the formula

$$\forall y \exists X\ \big(block(X) \wedge X \subseteq Z \wedge \forall x\ (x \in X \to y < x)\big),$$

but the size of the blocks in X is bounded, as stated by the formula

$$\neg \mathsf{U}X\ (block(X) \wedge X \subseteq Z).$$

Note that the set Z is infinite. This will play a role later on, when we talk about weak logics, which can only quantify over finite sets.

The class of languages of ω-words that can be defined in MSO+U is our first candidate for a new definition of "regular languages". It is also the largest class considered in this paper – it contains all the other classes that will be described below. By its very definition, the class is closed under union, complementation, projection, etc. The big problem is that we do not know if satisfiability is decidable for formulas of MSO+U over ω-words, although we conjecture it is.

Of course, decidable emptiness/satisfiability is very important if we want to talk about "regular languages". We try to attack this question by introducing automata models, some of which are described below. There will be the usual tradeoffs: nondeterministic automata are closed under projections (existential set quantifiers), while deterministic automata are closed under boolean operations.

We begin with the strongest automaton model, namely nondeterministic BS-automata, which were introduced in [3][1].

Nondeterministic BS-automata. A nondeterministic BS-automaton is defined like an NFA. The differences are: it does not have a set of accepting states, and it is equipped with a finite set C of counters, a counter update function and acceptance condition, as described below. The counter update function maps each transition to a finite, possibly empty, sequence of operations of the form

$$c := c + 1 \quad c := 0 \quad c := d \qquad \text{for } c, d \in C.$$

Let ρ be a run of the automaton over an input ω-word, as defined for nondeterministic automata on infinite words. The set of runs for a given input word is independent of the counters, counter update function and acceptance condition.

What are the counters used for? They are used to say when a run ρ is accepting. For a counter $c \in C$ and a word position $i \in \mathbb{N}$, we consider the number $val(\rho, c, i)$, which is the value of counter c after doing the first i transitions. (All counters start with zero.) These numbers are then examined by the acceptance condition, which talks about their assymptotic behavior. (This explains why nondeterministic BS-automata cannot describe patterns usually associated with counter automata, such as $a^n b^n$.) Specifically, the acceptance condition is a positive boolean combination of conditions of the three kinds below.

$$\limsup_i val(\rho, c, i) < \infty \qquad \liminf_i val(\rho, c, i) = \infty \qquad \text{“state } q \text{ appears infinitely often”}$$

The first kind of condition is called a *B-condition* (because it requires counter c to be bounded), the second kind of condition is called an *S-condition* (in [3], a number sequence converging to ∞ was called “strongly unbounded”), and the last kind of condition is called a *Büchi condition.*

Emptiness for nondeterministic BS-automata is decidable [3]. The emptiness procedure searches for something like the “lasso” that witnesses nonemptiness of a Büchi automaton. The notion of lasso for nondeterministic BS-automata is more complicated, and leads to a certain class of finitely representable infinite words, a class which extends the class of ultimately periodic words.

Consider the languages recognized by nondeterministic BS-automata. These languages are closed under union and intersection, thanks to the usual product construction. These languages are closed under projection (or existential set quantification), thanks to nondeterminism. These languages are also closed under a suitable definition of the quantifier U for languages, see [3]. If these languages were also closed under complement, then nondeterministic BS-automata would recognize all languages definable in MSO+U (and nothing more, since existence of an accepting run of a nondeterministic BS-automaton can be described in the logic).

Unfortunately, complementation fails. There is, however, a partial complementation result, which concerns two subclasses of nondeterministic BS-automata. An automaton

[1] For consistency of presentation, the definition given here is slightly modified from the one in [3]: the automata can move values between counters, and they can use Büchi acceptance conditions. These changes do not affect the expressive power.

that does not use S-conditions is called a *B-automaton*; an automaton that does not use B-conditions is called an *S-automaton.*

Theorem 1 ([3])**.** *The complement of a language recognized by a nondeterministic B-automaton is recognized by a nondeterministic S-automaton, and vice versa.*

The correspondence is effective: from a B-automaton we can compute an S-automaton for the complement, and vice versa. The proof of Theorem 1 is difficult, because it has to deal with nondeterministic automata. (Somewhat like complementation of nondeterministic automata on infinite trees in the proof of Rabin's theorem.) The technical aspects are similar to, but more general than, Kirsten's decidability proof [8] of the star height problem in formal language theory. In particular, it is not difficult to prove, using Theorem 1, that the star height problem is decidable.

Deterministic max-automata. As mentioned above, nondeterministic BS-automata are not closed under complement. A typical approach to the complementation problem is to consider deterministic automata; this is the approach described below, following [2].

A *deterministic max-automaton* is defined like a BS-automaton, with the following differences: a) it is deterministic; b) it has an additional counter operation $c := \max(d, e)$; and c) its acceptance condition is a boolean (not necessarily positive) combination of B-conditions. The max operation looks dangerous, since it seems to involve arithmetic. However, the counters are only tested for the limits, and this severely restricts the way max can be used. One can show that nondeterminism renders the max operation redundant, as stated by Theorem 2 below. (For deterministic automata, max is not redundant.)

Theorem 2 ([2])**.** *Every language recognized by a deterministic max-automaton is a boolean combination of languages recognized by nondeterministic B-automata.*

By Theorem 1, every boolean combination of languages recognized by nondeterministic B-automata is equivalent to a *positive* boolean combination of languages recognized by nondeterministic B-automata, and nondeterministic S-automata. Such a positive boolean combination is, in turn, recognized by a single nondeterministic BS-automaton, since these are closed under union and intersection. It follows that every deterministic max-automaton is equivalent to a nondeterministic BS-automaton. Since the equivalence is effective, we get an algorithm for deciding emptiness of deterministic max-automata. (A direct approach to deciding emptiness of deterministic max-automata is complicated by the max operation.)

So what is the point of deterministic max-automata?

The point is that they have good closure properties. (This also explains why the max operation is used. The version without max does not have the closure properties described below.) Since the automata are deterministic, and the acceptance condition is closed under boolean combinations, it follows that languages recognized by deterministic max-automata are closed under boolean combinations. What about the existential set quantifier? If we talk about set quantification like in MSO, where infinite sets are quantified, then the answer is no [2]; closure under existential set quantifiers is essentially equivalent to nondeterminism. However, it turns out that quantification over *finite* sets can be implemented by deterministic max-automata, which is stated by Theorem 3 below. The theorem refers to weak MSO+U, which is the fragment of MSO+U where the set quantifiers $\exists$ and $\forall$ are restricted to finite sets.

Theorem 3 ([2])**.** *Deterministic max-automata recognize exactly the languages that can be defined in weak MSO+*U*.*

Other deterministic automata. There is a natural dual automaton to a deterministic max-automaton, namely a *deterministic min-automaton*, see [4]. Instead of max this automaton uses min; instead of boolean combinations of B-conditions, it uses boolean combinations of S-conditions. While the duality is fairly clear on the automaton side, it is less clear on the logic side: we have defined only one new quantifier U, and this quantifier is already taken by max-automata, which capture exactly weak MSO+U.

The answer is to add a new quantifier R, which we call the *recurrence quantifier.* If quantification over infinite sets is allowed, the quantifier R can be defined in terms of U and vice versa; so we do not need to talk about the logic MSO+U+R. For weak MSO, the new quantifier is independent. So what does this new quantifier say? It says that the family of sets X satisfying $\varphi(X)$ contains infinitely many sets of the same finite size:

$$\mathsf{R}X\ \varphi(X) = \bigvee_{n\in\mathbb{N}} \exists_\infty X\ \big(\ \varphi(X) \quad \wedge \quad |X| = n\ \big).$$

If the quantifier U corresponds to the complement of the language L_B (it can say there are arbitrarily large blocks); the new quantifier R corresponds to the complement of the language L_S (it can say some block size appears infinitely often).

Theorem 4 ([4])**.** *Deterministic min-automata recognize exactly the languages that can be defined in weak MSO+*R*.*

The proof shares many similarities with the proof of Theorem 3. Actually, some of these similarities can be abstracted into a general framework on deterministic automata, which is the main topic of [4]. One result obtained from this framework, Theorem 5 below, gives an automaton model for weak MSO with both quantifiers U and R.

Theorem 5 ([4])**.** *Boolean combinations of deterministic min-automata and deterministic max-automata recognize exactly the languages that can be defined in weak MSO+*U*+*R*.*

The framework also works for different quantifiers, such as a perodicity quantifier (which binds a first-order variable x instead of a set variable X), defined as follows

$$\mathsf{P}x\ \varphi(x) = \text{the positions } x \text{ that satisfy } \varphi(x) \text{ are ultimately periodic.}$$

Closing remarks. Above, we have described several classes of languages of ω-words, defined by: the logics with new quantifiers and automata with counters. Each of the classes captures all the ω-regular languages, and more. Some of the models are more powerful, others have better closure properties; all describe languages that can reasonably be called "regular".

There is a lot of work to do on this topic. The case of trees is a natural candidate, some results on trees can be found in [6, 7]. Another question is about the algebraic theory of the new languages; similar questions but in the context of finite words were explored in [5].

References

[1] M. Bojańczyk. A Bounding Quantifier. In *Computer Science Logic*, pages 41–55, 2004.

[2] M. Bojańczyk. Weak MSO with the Unbounding Quantifier. In *Symposium on Theoretical Aspects of Computer Science*, pages 233–245, 2009.

[3] M. Bojańczyk and T. Colcombet. ω-Regular Expressions with Bounds. In *Logic in Computer Science*, pages 285–296, 2006.

[4] M. Bojańczyk and S. Toruńczyk. Deterministic Automata and Extensions of Weak MSO. In *Foundations of Software Technology and Theoretical Computer Science*, 2009.

[5] T. Colcombet. The Theory of Stabilisation Monoids and Regular Cost Functions. In *International Colloquium on Automata, Languages and Programming*, 2009.

[6] T. Colcombet and C. Löding. The Nondeterministic Mostowski Hierarchy and Distance-Parity Automata. In *International Colloquium on Automata, Languages and Programming* 2008: 398-409

[7] T. Colcombet and C. Löding. The Nesting-Depth of Disjunctive mu-calculus for Tree Languages and the Limitedness Problem. In *Computer Science Logic*, pages 416-430, 2008

[8] D. Kirsten. Distance desert automata and the star height problem. *Theoretical Informatics and Applications*, 39(3):455–511, 2005.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 17-32
www.stacs-conf.org

REFLECTIONS ON MULTIVARIATE ALGORITHMICS AND PROBLEM PARAMETERIZATION

ROLF NIEDERMEIER

Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany
E-mail address: `rolf.niedermeier@uni-jena.de`

Abstract. Research on parameterized algorithmics for NP-hard problems has steadily grown over the last years. We survey and discuss how parameterized complexity analysis naturally develops into the field of multivariate algorithmics. Correspondingly, we describe how to perform a systematic investigation and exploitation of the "parameter space" of computationally hard problems.

Algorithms and Complexity; Parameterized Algorithmics; Coping with Computational Intractability; Fixed-Parameter Tractability

1. Introduction

NP-hardness is an every-day obstacle for practical computing. Since there is no hope for polynomial-time algorithms for NP-hard problems, it is pragmatic to accept exponential-time behavior of solving algorithms. Clearly, an exponential growth of the running time is bad, but maybe affordable, if the combinatorial explosion is modest and/or can be confined to certain problem parameters. This line of research has been pioneered by Downey and Fellows' monograph "Parameterized Complexity" [24] (see [32, 57] for two more recent monographs). The number of investigations in this direction has steadily grown over the recent years. A core question herein is what actually "a" or "the" parameter of a computational problem is. The simple answer is that there are many reasonable possibilities to "parameterize a problem". In this survey, we review some aspects of this "art" of problem parameterization.[1] Moreover, we discuss corresponding research on multivariate algorithmics, the natural sequel of parameterized algorithmics when expanding to multidimensional parameter spaces.

We start with an example. The NP-complete problem Possible Winner for k-Approval is a standard problem in the context of voting systems. In the k-*approval* protocol, for a given set of candidates, each voter can assign a score of 1 to k of these candidates and the rest of the candidates receive score 0. In other words, each voter may linearly order the candidates; the "first" k candidates in this order score 1 and the remaining ones score 0. A winner of an election (where the input is a collection of votes) is a candidate who achieves the maximum total score. By simple counting this voting protocol can be

[1] In previous work [56, 57], we discussed the "art" of parameterizing problems in a less systematic way.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2495

evaluated in linear time. In real-world applications, however, a voter may only provide a partial order of the candidates: The input of POSSIBLE WINNER FOR k-APPROVAL is a set of partial orders on a set of candidates and a distinguished candidate d, and the question is whether there exists an extension for each partial order into a linear one such that d wins under the k-approval protocol. POSSIBLE WINNER FOR k-APPROVAL is NP-complete already in case of only two input votes when k is part of the input [10]. Moreover, for an unbounded number of votes POSSIBLE WINNER FOR 2-APPROVAL is NP-complete [7]. Hence, POSSIBLE WINNER FOR k-APPROVAL parameterized by the number v of votes as well as parameterized by k remains intractable. In contrast, the problem turns out to be fixed-parameter tractable when parameterized by the combined parameter (v, k) [6], that is, it can be solved in $f(v, k) \cdot \text{poly}$ time for some computable function f only depending on v and k (see Section 2 for more on underlying notions). In summary, this implies that to better understand and cope with the computational complexity of POSSIBLE WINNER FOR k-APPROVAL, we should investigate its parameterized (in)tractability with respect to various parameters and combinations thereof. Parameter combinations—this is what multivariate complexity analysis refers to—may be unavoidable to get fast algorithms for relevant special cases. In case of POSSIBLE WINNER FOR k-APPROVAL such an important special case is a small number of votes[2] together with a small value of k. Various problem parameters often come up very naturally. For instance, besides v and k, a further parameter here is the number c of candidates. Using integer linear programming, one can show that POSSIBLE WINNER FOR k-APPROVAL is fixed-parameter tractable with respect to the parameter c [10].

Idealistically speaking, multivariate algorithmics aims at a holistic approach to determine the "computational nature" of each NP-hard problem. To this end, one wants to find out which problem-specific parameters influence the problem's complexity in which quantitative way. Clearly, also combinations of several single parameters should be investigated. Some parameterizations may yield hardness even in case of constant values, some may yield polynomial-time solvability in case of constant values, and in the best case some may allow for fixed-parameter tractability results.[3] Hence, the identification of "reasonable" problem parameters is an important issue in multivariate algorithmics. In what follows, we describe and survey systematic ways to find interesting problem parameters to be exploited in algorithm design. This is part of the general effort to better understand and cope with computational intractability, culminating in the multivariate approach to computational complexity analysis.

2. A Primer on Parameterized and Multivariate Algorithmics

Consider the following two NP-hard problems from algorithmic graph theory. Given an undirected graph, compute a minimum-cardinality set of vertices that either cover all graph edges (this is VERTEX COVER) or dominate all graph vertices (this is DOMINATING SET). Herein, an edge e is *covered* by a vertex v if v is one of the two endpoints of e, and a vertex v is *dominated* by a vertex u if u and v are connected by an edge. By definition, every vertex dominates itself. The NP-hardness of both problems makes the search for

[2] There are realistic voting scenarios where the number of candidates is large and the number of voters is small. For instance, this is the case when a small committee decides about many applicants.

[3] For input size n and parameter value k, a running time of $O(n^k)$ would mean polynomial-time solvable for constant values of k whereas a running time of say $O(2^k n)$ would mean fixed-parameter tractability with respect to the parameter k, see Section 2 for more on this.

polynomial-time solving algorithms hopeless. How fast can we solve these two minimization problems in an exact way? Trying all possibilities, for an n-vertex graph in case of both problems we end up with an algorithm running in basically 2^n steps (times a polynomial), being infeasible for already small values of n. However, what happens if we only search for a size-at-most-k solution set? Trying all size-k subsets of the n-vertex set as solution candidates gives a straightforward algorithm running in $O(n^{k+2})$ steps. This is superior to the 2^n-steps algorithm for sufficiently small values of k, but again turns infeasible already for moderate k-values. Can we still do better? Yes, we can—but seemingly only for Vertex Cover. Whereas we do not know any notably more efficient way to solve Dominating Set [24, 20], in case of Vertex Cover a simple observation suffices to obtain a 2^k-step (times a polynomial) algorithm: Just pick any edge and branch the search for a size-k solution into the two possibilities of taking one of the two endpoints of this edge. One of them has to be in an optimal solution! Recurse (branching into two subcases) to find size-$(k-1)$ solutions for the remaining graphs where the already chosen vertex is deleted. In this way, one can achieve a search tree of size 2^k, leading to the stated running time. In summary, there is a simple 2^k-algorithm for Vertex Cover whereas there is only an $n^{O(k)}$-algorithm for Dominating Set. Clearly, this makes a huge difference in practical computing, although both algorithms can be put into the coarse category of "polynomial time for constant values of k". This categorization ignores that in the one case k influences the degree of the polynomial and in the other it does not—the categorization is too coarse-grained; a richer modelling is needed. This is the key contribution parameterized complexity analysis makes.

To better understand the different behavior of Vertex Cover and Dominating Set concerning their solvability in dependence on the *parameter* k (solution size) historically was one of the starting points of parameterized complexity analysis [24, 32, 57]. Roughly speaking, it deals with a "function battle", namely the typical question whether an $n^{O(k)}$-algorithm can be replaced by a significantly more efficient $f(k)$-algorithm where f is a computable function exclusively depending on k; in more general terms, this is the question for the fixed-parameter tractability (fpt) of a computationally hard problem. Vertex Cover is fpt, Dominating Set, classified as W[1]-hard (more precisely, W[2]-complete) by parameterized complexity theory, is very unlikely to be fpt. Intuitively speaking, a parameterized problem being classified as W[1]-hard with respect to parameter k means that it is as least as hard as computing a k-vertex clique in a graph. There seems to be no hope for doing this in $f(k) \cdot n^{O(1)}$ time for a computable function f.

More formally, parameterized complexity is a two-dimensional framework for studying the computational complexity of problems [24, 32, 57]. One dimension is the input size n (as in classical complexity theory), and the other one is the *parameter* k (usually a positive integer). A problem is called *fixed-parameter tractable* (fpt) if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function only depending on k. This means that when solving a problem that is fpt, the combinatorial explosion can be confined to the parameter. There are numerous algorithmic techniques for the design of fixed-parameter algorithms, including data reduction and kernelization [11, 41], color-coding [3] and chromatic coding [2], iterative compression [58, 40], depth-bounded search trees, dynamic programming, and several more [44, 60]. Downey and Fellows [24] developed a parameterized theory of computational complexity to show fixed-parameter intractability. The basic complexity class for fixed-parameter intractability is called W[1] and there is good reason to believe that W[1]-hard problems are not fpt [24, 32, 57]. Indeed, there is a whole complexity hierarchy FPT $\subseteq$

W[1] $\subseteq$ W[2] $\subseteq \ldots \subseteq$ XP, where XP denotes the class of parameterized problems that can be solved in polynomial time in case of constant parameter values. See Chen and Meng [22] for a recent survey on parameterized hardness and completeness. Indeed, the typical expectation for a parameterized problem is that it either is in FPT or is W[1]-hard but in XP or already is NP-hard for some constant parameter value.

In retrospective, the one-dimensional NP-hardness theory [34] and its limitations to offer a more fine-grained description of the complexity of exactly solving NP-hard problems led to the two-dimensional framework of parameterized complexity analysis. Developing further into multivariate algorithmics, the number of corresponding research challenges grows, on the one hand, by identifying meaningful different parameterizations of a single problem, and, on the other hand, by studying the combinations of single parameters and their impact on problem complexity. Indeed, multivariation is the continuing revolution of parameterized algorithmics, lifting the two-dimensional framework to a multidimensional one [27].

3. Ways to Parameter Identification

From the very beginning of parameterized complexity analysis the "standard parameterization" of a problem referred to the cost of the solution (such as the size of a vertex set covering all edges of a graph, see Vertex Cover). For graph-modelled problems, "structural" parameters such as treewidth (measuring the treelikeness of graphs) also have played a prominent role for a long time. As we try to make clear in the following, structural problem parameterization is an enormously rich field. It provides a key to better understand the "nature" of computational intractability. The ultimate goal is to quantitatively classify how parameters influence problem complexity. The more we know about these interactions, the more likely it becomes to master computational intractability.

Structural parameterization, in a very broad sense, is the major issue of this section. However, there is also more to say about parameterization by "solution quality" (solution cost herein being one aspect), which is discussed in the first subsection. This is followed by several subsections which can be interpreted as various aspects of structural parameterization. It is important to realize that it may often happen that different parameterization strategies eventually lead to the same parameter. Indeed, also the proposed strategies may overlap in various ways. Still, however, each of the subsequent subsections shall provide a fresh view on parameter identification.

3.1. Parameterizations Related to Solution Quality

The Idea. The classical and most often used problem parameter is the cost of the solution sought after. If the solution cost is large, then it makes sense to study the dual parameter (the cost of the elements *not* in the solution set) or above guarantee parameterization (the guarantee is the minimum cost every solution must have and the parameter measures the distance from this lower bound). Solution quality, however, also may refer to quality of approximation as parameter, or the "radius" of the search area in local search (a standard method to design heuristic algorithms where the parameter k determines the size of a k-local neighborhood searched).

Examples. To find a size-k vertex cover in an n-vertex graph is solvable in $O(1.28^k + kn)$ time [21], that is, Vertex Cover is fixed-parameter tractable. In contrast, finding a size-k dominating set is W[1]-hard. In case of Vertex Cover, the dual parameterization leads to searching for a size-$(n-k')$ vertex cover, where k' is the number of vertices not contained in the vertex cover. This problem is W[1]-hard with respect to the parameter k' [24]. Indeed, this problem is equivalent to finding a size-k' independent set of vertices in a graph. This means that the corresponding problems Vertex Cover and Independent Set are dual to each other.

Above guarantee parameterization was pioneered by Mahajan and Raman [49] studying the Maximum Satisfiability problem, noting that in every boolean formula in conjunctive normal form one can satisfy at least half of all clauses. Hence, an obvious parameterization (leading to fixed-parameter tractability) is whether one can satisfy at least $\lceil m/2 \rceil + k$ clauses of a formula in conjunctive normal form. Herein, m denotes the total number of clauses and the parameter is k, measuring the distance to the guaranteed threshold $\lceil m/2 \rceil$. There is recent progress on new techniques and results in this direction [50, 1]. A long-standing open problem is to determine the parameterized complexity of finding a size-$(\lceil n/4 \rceil + k)$ independent set in an n-vertex planar graph, parameterized by k.

Marx [53] surveyed many facets of the relationship between approximation and parameterized complexity. For instance, he discussed the issue of ratio-$(1+\epsilon)$ approximation (that is, polynomial-time approximation schemes (PTAS's)) parameterized by the quality of approximation measure $1/\epsilon$. The central question here is whether the degree of the polynomial of the running time depends on the parameter $1/\epsilon$ or not.

Khuller et al. [45] presented a fixed-parameter tractability result for k-local search (parameterized by k) for the Minimum Vertex Feedback Edge Set problem. In contrast, Marx [54] provided W[1]-hardness results for k-local search for the Traveling Salesman problem. Very recently, fixed-parameter tractability results for k-local search for planar graph problems have been reported [31].

Discussion. Parameterization by solution quality becomes a colorful research topic when going beyond the simple parameter "solution size." Above guarantee parameterization and k-local search parameterization still seem to be at early development stages. The connections of parameterization to polynomial-time approximation and beyond still lack a deep and thorough investigation [53].

3.2. Parameterization by Distance from Triviality

The Idea. Identify polynomial-time solvable special cases of the NP-hard problem under study. A "distance from triviality"-parameter then shall measure how far the given instance is away from the trivial (that is, polynomial-time solvable) case.

Examples. A classical example for "distance from triviality"-parameterization are width concepts measuring the similarity of a graph compared to a tree. The point is that many graph problems that are NP-hard on general graphs become easily solvable when restricted to trees. The larger the respective width parameter is, the less treelike the considered graph is. For instance, Vertex Cover and Dominating Set both become fixed-parameter tractable with respect to the treewidth parameter; see Bodlaender and Koster [12] for a survey. There are many more width parameters measuring the treelikeness of graphs, see Hliněný et al. [42] for a survey.

Besides measuring treewidth, alternatively one may also study the feedback vertex set number to measure the distance from a tree. Indeed, the feedback vertex set number of a graph is at least as big as its treewidth. Kratsch and Schweitzer [47] showed that the Graph Isomorphism problem is fixed-parameter tractable when parameterized by the feedback vertex set size; in contrast, this is open with respect to the parameter treewidth. A similar situation occurs when parameterizing the Bandwidth problem by the vertex cover number of the underlying graph [30].

Further examples for the "distance from triviality"-approach appear in the context of vertex-coloring of graphs [18, 51]. Here, for instance, coloring chordal graphs is polynomial-time solvable and the studied parameter measures how many edges to delete from a graph to make it chordal; this turned out to be fixed-parameter tractable [51]. Deiněko et al. [23] and Hoffman and Okamoto [43] described geometric "distance from triviality"-parameters by measuring the number of points inside the convex hull of a point set. A general view on "distance from triviality"-parameterization appears in Guo et al. [39].

Discussion. Measuring distance from triviality is a very broad and flexible way to generate useful parameterizations of intractable problems. It helps to better analyze the transition from polynomial- to exponential-time solvability.

3.3. Parameterization Based on Data Analysis

The Idea. With the advent of algorithm engineering, it has become clear that algorithm design and analysis for practically relevant problems should be part of a development cycle. Implementation and experiments with a base algorithm combined with standard data analysis methods provide insights into the structure of the considered real-world data which may be quantified by parameters. Knowing these parameters and their typical values then can inspire new solving strategies based on multivariate complexity analysis.

Examples. A very simple data analysis in graph problems would be to check the maximum vertex degree of the input graph. Many graph problems can be solved faster when the maximum degree is bounded. For instance, Independent Set is fixed-parameter tractable on bounded-degree graphs (a straightforward depth-bounded search tree does) whereas it is W[1]-hard on general graphs.

Song et al. [61] described an approach for the alignment of a biopolymer sequence (such as an RNA or a protein) to a structure by representing both the sequence and the structure as graphs and solving some subgraph problem. Observing the fact that for real-world instances the structure graph has small treewidth, they designed practical fixed-parameter algorithms based on the parameter treewidth. Refer to Cai et al. [19] for a survey on parameterized complexity and biopolymer sequence comparison.

A second example deals with finding dense subgraphs (more precisely, some form of clique relaxations) in social networks [55]. Here, it was essential for speeding up the algorithm and making it practically competitive that there were only relatively few hubs (that is, high-degree vertices) in the real-world graph. The corresponding algorithm engineering exploited this low parameter value.

Discussion. Parameterization by data analysis goes hand in hand with algorithm engineering and a *data-driven* algorithm design process. It combines empirical findings (that is, small parameter values measured in the input data) with rigorous theory building (provable fixed-parameter tractability results). This line of investigation is still underdeveloped in parameterized and multivariate algorithmics but is a litmus test for the practical relevance and impact on applied computing.

3.4. Parameterizations Generated by Deconstructing Hardness Proofs

The Idea. Look at the (many-one) reductions used to show a problem's NP-hardness. Check whether certain quantities (that is, parameters) are assumed to be unbounded in order to make the reduction work. Parameterize by these quantities. It is important to note that this approach naturally extends to deconstructing W[1]-hardness proofs; here the goal is to find additional parameters to achieve fixed-parameter tractability results.

Examples. Recall our introductory example with Possible Winner for k-Approval. From the corresponding NP-hardness proofs it follows that this problem is NP-hard when either the number of votes v is a constant (but k is unbounded) or k is a constant (but v is unbounded) [7, 10], whereas it becomes fixed-parameter tractable when parameterized by both k and v [6].

A second example, where the deconstruction approach is also systematically explained, refers to the NP-hard Interval Constrained Coloring problem [46]. Looking at a known NP-hardness proof [4], one may identify several quantities being unbounded in the NP-hardness reduction; this was used to derive several fixed-parameter tractability results [46]. In contrast, a recent result showed that the quantity "number k of colors" alone is not useful as a parameter in the sense that the problem remains NP-hard when restricted to instances with only three colors [15]. Indeed, Interval Constrained Coloring offers a multitude of challenges for multivariate algorithmics, also see Subsection 4.3.

Discussion. Deconstructing intractability relies on the close study of the available hardness proofs for an intractable problem. This means to strive for a full understanding of the current state of knowledge about a problem's computational complexity. Having identified quantities whose unboundedness is essential for the hardness proofs then can trigger the search for either stronger hardness or fixed-parameter tractability results.

3.5. Parameterization by Dimension

The Idea. The dimensionality of a problem plays an important role in computational geometry and also in fields such as databases and query optimization (where the dimension number can be the number of attributes of a stored object). Hence, the dimension number and also the "range of values of each dimension" are important for assessing the computational complexity of multidimensional problems.

Examples. Cabello et al. [16] studied the problem to decide whether two n-point sets in d-dimensional space are congruent, a fundamental problem in geometric pattern matching. Brass and Knauer [13] conjectured that this problem is fixed-parameter tractable with respect to the parameter d. However, deciding whether a set is congruent to a subset of another set is shown to be W[1]-hard with respect to d [16]. An other example appears in the context of geometric clustering. Cabello et al. [17] showed that the RECTILINEAR 3-CENTER problem is fixed-parameter tractable with respect to the dimension of the input point set whereas RECTILINEAR k-CENTER for $k \geq 4$ and EUCLIDEAN k-CENTER for $k \geq 2$ are W[1]-hard with respect to the dimension parameter. See Giannopoulos et al. [35, 36] for more on the parameterized complexity of geometric problems.

The CLOSEST STRING problem is of different "dimension nature". Here, one is given a set of k strings of same length and the task is to find a string which minimizes the maximum Hamming distance to the input strings. The two dimensions of this problem are string length (typically large) and number k of strings (typically small). It was shown that CLOSEST STRING is fixed-parameter tractable with respect to the "dimension parameter" k [38], whereas fixed-parameter tractability with respect to the string length is straightforward in the case of constant-size input alphabets; also see Subsection 4.1.

Discussion. Incorporating dimension parameters into investigations is natural and the parameter values and ranges usually can easily be derived from the applications. The dimension alone, however, usually seems to be a "hard parameter" in terms of fixed-parameter tractability; so often the combination with further parameters might be unavoidable.

3.6. Parameterization by Averaging Out

The Idea. Assume that one is given a number of objects and a distance measure between them. In median or consensus problems, the goal is to find an object that minimizes the sum of distances to the given objects. Parameterize by the average distance to the goal object or the average distance between the input objects. In graph problems, the average vertex degree could for instance be an interesting parameter.

Examples. In the CONSENSUS PATTERNS problem, for given strings $s_1, \ldots, s_k$ one wants to find a string s of some specified length such that each s_i, $1 \leq i \leq k$, contains a substring such that the average of the distances of s to these k substrings is minimized. Marx [52] showed that CONSENSUS PATTERNS is fixed-parameter tractable with respect to this average distance parameter.

In the CONSENSUS CLUSTERING problem, one is given a set of n partitions $C_1, \ldots, C_n$ of a base set S. In other words, every partition of the base set is a clustering of S. The goal is to find a partition C of S that minimizes the sum $\sum_{i=1}^{n} d(C, C_i)$, where the distance function d measures how similar two clusters are by counting the "differently placed" elements of S. In contrast to CONSENSUS PATTERNS, here the parameter "average distance between two input partitions" has been considered and led to fixed-parameter tractability [9]. Thus, the higher the degree of average similarity between input objects is, the faster one finds the desired median object.

Discussion. The average parameterization for Consensus Patterns directly relates to the solution quality whereas the one for Consensus Clustering relates to the structure of the input. In the latter case, the described example showed that one can deal with “outliers” having high distance to the other objects. Measuring the average distance between the input objects means to determine their degree of average similarity. This structural parameter value may be quickly computed in advance, making it easy to forecast the performance of the corresponding fixed-parameter algorithm.

4. Three Case Studies

In the preceding section, we focussed on various ways to single out various interesting problem parameterizations. In what follows, we put emphasis on the multivariate aspects of complexity analysis related to (combining) different parameterizations of one and the same problem. To this end, we study three NP-hard problems that nicely exhibit various relevant features of multivariate algorithmics.

4.1. Closest String

The NP-hard Closest String problem is to find a length-L string that minimizes the maximum Hamming distance to a given set of k length-L strings. The problem arises in computational biology (motif search in strings) and coding theory (minimum radius problem).

Known Results. What are natural parameterizations here? First, consider the number k of input strings. Using integer linear programming results, fixed-parameter tractability with respect to k can be derived [38]. This result is of theoretical interest only due to a huge combinatorial explosion. Second, concerning the parameter string length L, for strings over alphabet Σ we obviously only need to check all $|\Sigma|^L$ candidates for the closest string and choose a best one, hence fixed-parameter tractability with respect to L follows for constant-size alphabets. More precisely, Closest String is fixed-parameter tractable with respect to the combined parameter $(|\Sigma|, L)$. Finally, recall that the goal is to minimize the maximum distance d; thus, d is a natural parameter as well, being small (say values below 10) in biological applications. Closest String is also shown to be fixed-parameter tractable with respect to d by designing a search tree of size $(d+1)^d$ [38]. A further fixed-parameter algorithm with respect to the combined parameter $(|\Sigma|, d)$ has a combinatorial explosion of the form $(|\Sigma|-1)^d \cdot 2^{4d}$ [48], which has recently been improved to $(|\Sigma|-1)^d \cdot 2^{3.25d}$ [62]. For small alphabet size these results improve on the $(d+1)^d$-search tree algorithm. There are also several parameterized complexity results on the more general Closest Substring and further related problems [29, 37, 52, 48, 62].

Discussion. Closest String carries four obvious parameters, namely the number k of input strings, the string length L, the alphabet size $|\Sigma|$, and the solution distance d. A corresponding multivariate complexity analysis still faces several open questions with respect to making solving algorithms more practical. For instance, it would be interesting to see whether the (impractical) fixed-parameter tractability result for parameter k can be improved when adding further parameters. Moreover, it would be interesting to identify further structural string parameters that help to gain faster algorithms, perhaps in combination with known parameterizations. This is of particular importance for the more general and harder Closest Substring problem.

Data analysis has indicated small d- and k-values in biological applications. Interesting polynomial-time solvable instances would help to find "distance from triviality"-parameters. Closest String remains NP-hard for binary alphabets [33]; a systematic intractability deconstruction appears desirable. Closest String has the obvious two dimensions k and L, where k is typically much smaller than L. Parameterization by "averaging out" is hopeless for Closest String since one can easily many-one reduce an arbitrary input instance to one with constant average Hamming distance between input strings: just add a sufficiently large number of identical strings. Altogether, the multivariate complexity nature of Closest String is in many aspects unexplored.

4.2. Kemeny Score

The Kemeny Score problem is to find a consensus ranking of a given set of votes (that is, permutations) over a given set of candidates. A *consensus ranking* is a permutation of the candidates that minimizes the sum of "inversions" between this ranking and the given votes. Kemeny Score plays an important role in rank aggregation and multi-agent systems; due to its many nice properties, it is considered to be one of the most important preference-based voting systems.

Known Results. Kemeny Score is NP-hard already for four votes [25, 26], excluding hope for fixed-parameter tractability with respect to the parameter "number of votes". In contrast, the parameter "number of candidates" c trivially leads to fixed-parameter tractability by simply checking all possible $c!$ permutations that may constitute the consensus ranking. Using a more clever dynamic programming approach, the combinatorial explosion can be lowered to 2^c [8]. A different natural parameterization is to study what happens if the votes have high pairwise average similarity. More specifically, this means counting the number of inversions between each pair of votes and then taking the average over all pairs. Indeed, the problem is also fixed-parameter tractable with respect to this similarity value s, the best known algorithm currently incurring a combinatorial explosion of 4.83^s [59]. Further natural parameters are the sum of distances of the consensus ranking to input votes (that is, the Kemeny score) or the range of positions a candidate takes within a vote [8]. Other than for the pairwise distance parameter, where both the maximum and the average version lead to fixed-parameter tractability [8, 59], for the range parameter only the maximum version does whereas the problem becomes NP-hard already for an average range value of 2. [8]. Simjour [59] also studied the interesting parameter "Kemeny score divided by the number of candidates" and also showed fixed-parameter tractability in this case. There are more general problem versions that allow ties within the votes. Some fixed-parameter tractability results also have been achieved here [8, 9].

Discussion. Kemeny Score is an other example for a problem carrying numerous "obvious" parameters. Most known results, however, are with respect to two-dimensional complexity analysis (that is, parameterization by a single parameter), lacking the extension to a multivariate view.

First data analysis studies on ranking data [14] indicate the practical relevance of some of the above parameterizations. Average pairwise distance may be also considered as a straightforward "distance from triviality"-measure since average distance 0 means that all input votes are equal. The same holds true for the range parameter. Again, known intractability deconstruction for KEMENY SCORE just refers to looking at the NP-hardness result of Dwork et al. [25, 26], implying hardness already for a constant number of votes. A more fine-grained intractability deconstruction is missing. KEMENY SCORE can be seen as a two-dimensional problem. One dimension is the number of votes and the other dimension is number of candidates; however, only the latter leads to fixed-parameter tractability. In this context, the novel concept of "partial kernelization" has been introduced [9]. To the best of our knowledge, KEMENY SCORE has been the first example for a systematic approach to average parameterization [8, 9]. As for CLOSEST STRING, a multidimensional analysis of the computational complexity of KEMENY SCORE remains widely open.

4.3. Interval Constrained Coloring

In the NP-hard INTERVAL CONSTRAINED COLORING problem [4, 5] (arising in automated mass spectrometry in biochemistry) one is given a set of m integer intervals in the range 1 to r and a set of m associated multisets of colors (specifying for each interval the colors to be used for its elements), and one asks whether there is a "consistent" coloring for all integer points from $\{1, \dots, r\}$ that complies with the constraints specified by the color multisets.

Known Results. INTERVAL CONSTRAINED COLORING remains NP-hard even in case of only three colors [15]. Deconstructing the original NP-hardness proof due to Althaus et al. [4] and taking into account the refined NP-hardness proof of Byrka et al. [15], the following interesting parameters have been identified [46]:

- interval range,
- number of intervals,
- maximum interval length,
- maximum cutwidth with respect to overlapping intervals,
- maximum pairwise interval overlap, and
- maximum number of different colors in the color multisets.

All these quantities are assumed to be unbounded in the NP-hardness reduction due to Althaus et al. [4]; this immediately calls for a parameterized investigation. Several fixed-parameter tractability results have been achieved for single parameters and parameter pairs, leaving numerous open questions [46]. For instance, the parameterized complexity with respect to the parameter "number of intervals" is open, whereas INTERVAL CONSTRAINED COLORING is fixed-parameter tractable with respect to the parameter "interval length". Combining the parameters "number of colors" and "number of intervals" though, one achieves fixed-parameter tractability. In summary, many multidimensional parameterizations remain unstudied.

Discussion. The case of INTERVAL CONSTRAINED COLORING gives a prime example for deconstruction of intractability and the existence of numerous relevant parameterizations. There are a few known fixed-parameter tractability results, several of them calling for improved algorithms. Checking "all" reasonable parameter combinations and constellations could easily make an interesting PhD thesis.

The biological data often contain only three colors; the corresponding NP-hardness result [15] shows that this alone is not a fruitful parameter—combination with other parameters is needed (such as the interval range [46]). Moreover, observations on biological data indicate a small number of lengthy intervals, motivating a further parameterization possibility. Instances with only two colors or cutwidth two are "trivial" in the sense that (nontrivial) polynomial-time algorithms have been developed to solve these instances [4, 46]. Unfortunately, in both cases a parameter value of three already yields NP-hardness. The two natural dimensions of the problem are given by the interval range and the number of intervals, both important parameters. Average parameterization has not been considered yet. In summary, Interval Constrained Coloring might serve as a "model problem" for studying many aspects of multivariate algorithmics.

5. Conclusion with Six Theses on Multivariate Algorithmics

We described a number of possibilities to derive meaningful "single" parameterizations. Typically, not every such parameter will allow for fixed-parameter tractability results. Assume that a problem is W[1]-hard with respect to a parameter k (or even NP-hard for constant values of k). Then this calls for studying whether the problem becomes tractable when adding a further parameter k', that is, asking the question whether the problem is fixed-parameter tractable with respect to the (combined) parameter (k, k'). Moreover, even if a problem is classified to be fixed-parameter tractable with respect to a parameter k, this still can be practically useless. Hence, introducing a second parameter may open the route to practical fixed-parameter algorithms. Altogether, in its full generality such a "problem processing" forms the heart of multivariate algorithmics.

Fellows et al. [28] proposed to study the "complexity ecology of parameters". For the ease of presentation restricting the discussion to graph problems, one may build "complexity matrices" where both rows and columns represent certain parameters such as treewidth, bandwidth, vertex cover number, domination number, and so on. The corresponding values deliver structural information about the input graph. Then, a matrix entry in row x and column y represents a question of the form "how hard is it to compute the quantity represented by column y when parameterized by the quantity represented by x?". For example, it is easy to see that the domination number can be computed by a fixed-parameter algorithm using the parameter vertex cover number. Obviously, there is no need to restrict such considerations to two-dimensional matrices, thus leading to a full-flavored multivariate algorithmics approach.

After all, a multivariate approach may open Pandora's box by generating a great number of questions regarding the influence and the interrelationship between parameters in terms of computational complexity. With the tools provided by parameterized and multivariate algorithmics, the arising questions yield worthwhile research challenges. Indeed, to better understand important phenomena of computational complexity, there seems to be no way to circumvent such a "massive analytical attack" on problem complexity. Opening Pandora's box, however, is not hopeless because multivariate algorithmics can already rely on numerous tools available from parameterized complexity analysis.

There is little point in finishing this paper with a list of open questions—basically every NP-hard problem still harbors numerous challenges in terms of multivariate algorithmics. Indeed, multivariation is a horn of plenty concerning practically relevant and theoretically appealing opportunities for research. Instead, we conclude with six claims and conjectures concerning the future of (multivariate) algorithmics.

Thesis 1: Problem parameterization is a pervasive and ubiquitous tool in attacking intractable problems. A theory of computational complexity neglecting parameterized and multivariate analysis is incomplete.

Thesis 2: Multivariate algorithmics helps in gaining a more fine-grained view on polynomial-time solvable problems, also getting in close touch with adaptive algorithms.[4]

Thesis 3: Multivariate algorithmics can naturally incorporate approximation algorithms, relaxing the goal of exact to approximate solvability.

Thesis 4: Multivariate algorithmics is a "systems approach" to explore the nature of computational complexity. In particular, it promotes the development of meta-algorithms that first estimate various parameter values and then choose the appropriate algorithm to apply.

Thesis 5: Multivariate algorithmics helps to significantly increase the impact of Theoretical Computer Science on practical computing by providing more expressive statements about worst-case complexity.

Thesis 6: Multivariate algorithmics is an ideal theoretical match for algorithm engineering, both areas mutually benefiting from and complementing each other.

Acknowledgments. I am grateful to Nadja Betzler, Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Dániel Marx, Hannes Moser, Johannes Uhlmann, and Mathias Weller for constructive and insightful feedback on earlier versions of this paper.

References

[1] N. Alon, G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. Solving MAX-r-SAT above a tight lower bound. In *Proc. 21st SODA*. ACM/SIAM, 2010.

[2] N. Alon, D. Lokshtanov, and S. Saurabh. Fast FAST. In *Proc. 36th ICALP*, volume 5555 of *LNCS*, pages 49–58. Springer, 2009.

[3] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[4] E. Althaus, S. Canzar, K. Elbassioni, A. Karrenbauer, and J. Mestre. Approximating the interval constrained coloring problem. In *Proc. 11th SWAT*, volume 5124 of *LNCS*, pages 210–221. Springer, 2008.

[5] E. Althaus, S. Canzar, M. R. Emmett, A. Karrenbauer, A. G. Marshall, A. Meyer-Baese, and H. Zhang. Computing H/D-exchange speeds of single residues from data of peptic fragments. In *Proc. 23rd SAC '08*, pages 1273–1277. ACM, 2008.

[6] N. Betzler. On problem kernels for possible winner determination under the k-approval protocol. 2009.

[7] N. Betzler and B. Dorn. Towards a dichotomy of finding possible winners in elections based on scoring rules. In *Proc. 34th MFCS*, volume 5734 of *LNCS*, pages 124–136. Springer, 2009.

[8] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier, and F. A. Rosamond. Fixed-parameter algorithms for Kemeny scores. *Theor. Comput. Sci.*, 410(45):4454–4570, 2009.

[9] N. Betzler, J. Guo, C. Komusiewicz, and R. Niedermeier. Average parameterization and partial kernelization for computing medians. In *Proc. 9th LATIN*, LNCS. Springer, 2010.

[10] N. Betzler, S. Hemmann, and R. Niedermeier. A multivariate complexity analysis of determining possible winners given incomplete votes. In *Proc. 21st IJCAI*, pages 53–58, 2009.

[11] H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proc. 4th IWPEC*, volume 5917 of *LNCS*, pages 17–37. Springer, 2009.

[4] For instance, an adaptive sorting algorithm takes advantage of existing order in the input, with its running time being a function of the disorder in the input.

[12] H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comp. J.*, 51(3):255–269, 2008.

[13] P. Brass and C. Knauer. Testing the congruence of d-dimensional point sets. *Int. J. Comput. Geometry Appl.*, 12(1–2):115–124, 2002.

[14] R. Bredereck. Fixed-parameter algorithms for computing Kemeny scores—theory and practice. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany, 2009.

[15] J. Byrka, A. Karrenbauer, and L. Sanità. The interval constrained 3-coloring problem. In *Proc. 9th LATIN*, LNCS. Springer, 2010.

[16] S. Cabello, P. Giannopoulos, and C. Knauer. On the parameterized complexity of d-dimensional point set pattern matching. *Inf. Process. Lett.*, 105(2):73–77, 2008.

[17] S. Cabello, P. Giannopoulos, C. Knauer, D. Marx, and G. Rote. Geometric clustering: fixed-parameter tractability and lower bounds with respect to the dimension. *ACM Transactions on Algorithms*, 2009. To appear. Preliminary version at *SODA 2008*.

[18] L. Cai. Parameterized complexity of vertex colouring. *Discrete Appl. Math.*, 127(1):415–429, 2003.

[19] L. Cai, X. Huang, C. Liu, F. A. Rosamond, and Y. Song. Parameterized complexity and biopolymer sequence comparison. *Comp. J.*, 51(3):270–291, 2008.

[20] J. Chen, B. Chor, M. Fellows, X. Huang, D. W. Juedes, I. A. Kanj, and G. Xia. Tight lower bounds for certain parameterized NP-hard problems. *Inform. and Comput.*, 201(2):216–231, 2005.

[21] J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for Vertex Cover. In *Proc. 31st MFCS*, volume 4162 of *LNCS*, pages 238–249. Springer, 2006.

[22] J. Chen and J. Meng. On parameterized intractability: Hardness and completeness. *Comp. J.*, 51(1):39–59, 2008.

[23] V. G. Deiněko, M. Hoffmann, Y. Okamoto, and G. J. Woeginger. The traveling salesman problem with few inner points. *Oper. Res. Lett.*, 34(1):106–110, 2006.

[24] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[25] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *Proc. 10th WWW*, pages 613–622, 2001.

[26] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation revisited, 2001. Manuscript.

[27] M. Fellows. Towards fully multivariate algorithmics: Some new results and directions in parameter ecology. In *Proc. IWOCA*, volume 5874 of *LNCS*, pages 2–10. Springer, 2009.

[28] M. Fellows, D. Lokshtanov, N. Misra, M. Mnich, F. Rosamond, and S. Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory Comput. Syst.*, 45:822–848, 2009.

[29] M. R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of motif search problems. *Combinatorica*, 26(2):141–167, 2006.

[30] M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In *Proc. 19th ISAAC*, volume 5369 of *LNCS*, pages 294–305. Springer, 2008.

[31] M. R. Fellows, F. A. Rosamond, F. V. Fomin, D. Lokshtanov, S. Saurabh, and Y. Villanger. Local search: Is brute-force avoidable? In *Proc. 21st IJCAI*, pages 486–491, 2009.

[32] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer, 2006.

[33] M. Frances and A. Litman. On covering problems of codes. *Theory Comput. Syst.*, 30(2):113–119, 1997.

[34] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[35] P. Giannopoulos, C. Knauer, and G. Rote. The parameterized complexity of some geometric problems in unbounded dimension. In *Proc. 4th IWPEC*, volume 5917 of *LNCS*, pages 198–209. Springer, 2009.

[36] P. Giannopoulos, C. Knauer, and S. Whitesides. Parameterized complexity of geometric problems. *Comp. J.*, 51(3):372–384, 2008.

[37] J. Gramm, J. Guo, and R. Niedermeier. Parameterized intractability of distinguishing substring selection. *Theory Comput. Syst.*, 39(4):545–560, 2006.

[38] J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for Closest String and related problems. *Algorithmica*, 37(1):25–42, 2003.

[39] J. Guo, F. Hüffner, and R. Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proc. 1st IWPEC*, volume 3162 of *LNCS*, pages 162–173. Springer, 2004.

[40] J. Guo, H. Moser, and R. Niedermeier. Iterative compression for exactly solving NP-hard minimization problems. In *Algorithmics of Large and Complex Networks*, volume 5515 of *LNCS*, pages 65–80. Springer, 2009.

[41] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

[42] P. Hliněný, S. Oum, D. Seese, and G. Gottlob. Width parameters beyond tree-width and their applications. *Comp. J.*, 51(3):326–362, 2008.

[43] M. Hoffmann and Y. Okamoto. The minimum weight triangulation problem with few inner points. *Comput. Geom.*, 34(3):149–158, 2006.

[44] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *Comp. J.*, 51(1):7–25, 2008.

[45] S. Khuller, R. Bhatia, and R. Pless. On local search and placement of meters in networks. *SIAM J. Comput.*, 32(2):470–487, 2003.

[46] C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Deconstructing intractability—a case study for interval constrained coloring. In *Proc. 20th CPM*, volume 5577 of *LNCS*, pages 207–220. Springer, 2009.

[47] S. Kratsch and P. Schweitzer. Graph isomorphism parameterized by feedback vertex set number is fixed-parameter tractable. 2009.

[48] B. Ma and X. Sun. More efficient algorithms for closest string and substring problems. In *Proc. 12th RECOMB*, volume 4955 of *LNCS*, pages 396–409. Springer, 2008.

[49] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *J. Algorithms*, 31(2):335–354, 1999.

[50] M. Mahajan, V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. *J. Comput. System Sci.*, 75(2):137–153, 2009.

[51] D. Marx. Parameterized coloring problems on chordal graphs. *Theor. Comput. Sci.*, 351(3):407–424, 2006.

[52] D. Marx. Closest substring problems with small distances. *SIAM J. Comput.*, 38(4):1382–1410, 2008.

[53] D. Marx. Parameterized complexity and approximation algorithms. *Comp. J.*, 51(1):60–78, 2008.

[54] D. Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008.

[55] H. Moser, R. Niedermeier, and M. Sorge. Algorithms and experiments for clique relaxations—finding maximum s-plexes. In *Proc. 8th SEA*, volume 5526 of *LNCS*, pages 233–244. Springer, 2009.

[56] R. Niedermeier. Ubiquitous parameterization—invitation to fixed-parameter algorithms. In *Proc. 29th MFCS*, volume 3153 of *LNCS*, pages 84–103. Springer, 2004.

[57] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[58] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

[59] N. Simjour. Improved parameterized algorithms for the Kemeny aggregation problem. In *Proc. 4th IWPEC*, volume 5917 of *LNCS*, pages 312–323. Springer, 2009.

[60] C. Sloper and J. A. Telle. An overview of techniques for designing parameterized algorithms. *Comp. J.*, 51(1):122–136, 2008.

[61] Y. Song, C. Liu, X. Huang, R. L. Malmberg, Y. Xu, and L. Cai. Efficient parameterized algorithms for biopolymer structure-sequence alignment. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 3(4):423–432, 2006.

[62] L. Wang and B. Zhu. Efficient algorithms for the closest string and distinguishing string selection problems. In *Proc. 3rd FAW*, volume 5598 of *LNCS*, pages 261–270. Springer, 2009.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 33-34
www.stacs-conf.org

MATHEMATICS, CRYPTOLOGY, SECURITY

JACQUES STERN [1]

[1] Professor, Department of Computer Science, Ecole normale suprieure,
Chairman, Agence nationale de la recherche
E-mail address: Jacques.Stern@ens.fr

Abstract. In this talk, I will review some of the work performed by the research community in cryptology and security since the invention of public key cryptography by Diffie and Hellman in 1976. This community has developped many challenging lines of research. I will only focus on some of these, and moreover I will adopt an extremely specific perspective: for each chosen example, I will try to trace the original mathematics that underly the methods in use.

Over the years, maybe due to my original training as a mathematician, I have come to consider that linking recent advances and challenges in cryptology and security to the work of past mathematicians is indeed fascinating.

The range of examples will span both theory and practice: I will show that the celebrated RSA algorithm is intimately connected to mathematics that go back to the middle of the XVIIIth century. I will also cover alternatives to RSA, the method of "provable security", as well as some aspects of the security of electronic payments.

Key words and phrases: Mathematics, Cryptology, Security.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2441

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 35-46
www.stacs-conf.org

LARGE-GIRTH ROOTS OF GRAPHS

ANNA ADAMASZEK[1] AND MICHAŁ ADAMASZEK[2]

[1] Department of Computer Science and DIMAP,
University of Warwick, Coventry, CV4 7AL, UK
E-mail address: annan@mimuw.edu.pl

[2] Warwick Mathematics Institute and DIMAP,
University of Warwick, Coventry, CV4 7AL, UK
E-mail address: aszek@mimuw.edu.pl

Abstract. We study the problem of recognizing graph powers and computing roots of graphs. We provide a polynomial time recognition algorithm for r-th powers of graphs of girth at least $2r+3$, thus improving a bound conjectured by Farzad et al. (STACS 2009). Our algorithm also finds all r-th roots of a given graph that have girth at least $2r+3$ and no degree one vertices, which is a step towards a recent conjecture of Levenshtein that such root should be unique. On the negative side, we prove that recognition becomes an NP-complete problem when the bound on girth is about twice smaller. Similar results have so far only been attempted for $r = 2, 3$.

1. Introduction

All graphs in this paper are simple, undirected and connected. If H is a graph, its *r-th power* $G = H^r$ is the graph on the same vertex set such that two distinct vertices are adjacent in G if their distance in H is at most r. We also call H the *r-th root* of G.

There are some problems naturally related to graph powers and graph roots. Suppose $\mathcal{P}$ is a class of graphs (possibly consisting of all graphs), r is an integer and G is an arbitrary graph. The questions we ask are:

- *The recognition problem*: Is G an r-th power of some graph from $\mathcal{P}$? Formally, we define a family of decision problems:

  Problem. r-TH-POWER-OF-$\mathcal{P}$-GRAPH
  Instance. A graph G.
  Question. Is $G = H^r$ for some graph $H \in \mathcal{P}$?
- *The r-th root problem*: Find some/all r-th roots of G which belong to $\mathcal{P}$.
- *The unique reconstruction problem*: Is the r-th root of G in $\mathcal{P}$ (if any) unique?

1998 ACM Subject Classification: G.2.2 Graph algorithms, F.2.2 Analysis of algorithms and problem complexity.

Key words and phrases: Graph roots, Graph powers, NP-completeness, Recognition algorithms.

Research supported by the Centre for Discrete Mathematics and its Applications (DIMAP), EPSRC award EP/D063191/1.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2442

The above problems have been investigated for various graph classes $\mathcal{P}$. There exist characterizations of squares [15] and higher powers [3] of graphs, but they are not computationally efficient. Motwani and Sudan [14] proved the NP-completeness of recognizing graph squares and Lau [8] extended this to cubes of graphs. Motwani and Sudan [14] suggested that recognizing squares of bipartite graphs is also likely to be NP-complete. This was disproved by Lau [8], who gave a polynomial time algorithm that recognizes squares of bipartite graphs and counts the bipartite square roots of a given graph. Apparently the first proof that r-TH-POWER-OF-GRAPH and r-TH-POWER-OF-BIPARTITE-GRAPH are NP-complete for any $r \geq 3$ was recently announced in [10].

Considerable attention has been given to tree roots of graphs, which are quite well understood and can be computed efficiently, see Lin and Skiena [13], Kearney and Corneil [6] and Chang, Ko and Lu [2] who give a linear time algorithm for the r-th tree root of a given graph. Such a root need not be unique, not even up to isomorphism, so the difficulty lies in making consistent choices while constructing a root. Many techniques for computing tree roots rely on some sort of correspondence between vertex neighbourhoods in T and maximal cliques in T^p. We are going to use the computation of an r-th tree root of a graph as a black-box in our algorithms.

There has also been some work on the complexity of r-TH-POWER-OF-$\mathcal{P}$-GRAPH for such classes $\mathcal{P}$ as chordal graphs, split graphs and proper interval graphs [9] and for directed graphs and their powers [7].

In this work we address the above problems for another large family of graphs, namely graphs with no short cycles. Recall that the *girth* of a graph is the length of its shortest cycle (or ∞ for a tree). For convenience we shall denote by $\mathcal{GIRTH}_{\geq g}$ the class of all graphs of girth at least g, and by $\mathcal{GIRTH}^{+}_{\geq g}$ its subclass consisting of graphs with no vertices of degree one (which we call *leaves*). These classes of graphs make a convenient setting for graph roots because of the possible uniqueness results outlined below.

By [4] the recognition of squares of $\mathcal{GIRTH}_{\geq 4}$-graphs is NP-complete, while squares of $\mathcal{GIRTH}_{\geq 6}$-graphs can be recognized in polynomial time. The techniques of recognition (in this, and some other cases) include imposing some additional, local piece of information about the square root (like the existence of a certain edge) such that the root can then be reconstructed uniquely by expanding this data to the neighbouring vertices and eventually to the whole graph. Here we also exploit this idea.

For $r \geq 3$ no complexity-theoretic results have been known, but there is some very interesting work on the uniqueness of the roots. Precisely, Levenshtein et al. [12] proved that if G has a square root H in the class $\mathcal{GIRTH}^{+}_{\geq 7}$, then H is unique[1]. The same statement was extended in [11] to r-th roots in $\mathcal{GIRTH}^{+}_{\geq 2r+2\lceil (r-1)/4\rceil+1}$, using a characterization of the neighbourhood of a vertex as the unique set satisfying a list of properties expressed in terms of the r-th power of the graph. The main conjecture in this area remains unresolved:

Conjecture 1.1 (Levenshtein, [11])**.** If a graph G has an r-th root H in $\mathcal{GIRTH}^{+}_{\geq 2r+3}$, then H is unique in that class.

The value of $g = 2r + 3$ is best possible, as witnessed by the cycle C_{2r+2}, which cannot be uniquely reconstructed from its r-th power. The best result towards Conjecture 1.1 is

[1] It is not possible to obtain uniqueness if the vertices of degree one are allowed, hence this technical restriction. See [12] for details.

that the number of roots H under consideration is at most $\delta(G)$ (the minimal vertex degree in G, [11]), but its proof yields only exponential time r-th root and recognition algorithms.

At the same time Farzad et al. made a conjecture about recognizing powers of graphs of lower-bounded girth:

Conjecture 1.2 (Farzad et al., [4])**.** The problem r-TH-POWER-OF-$\mathcal{GIRTH}_{\geq 3r-1}$-GRAPH can be solved in polynomial time.

Our contribution. Our first result gives an efficient reconstruction algorithm in Levenshtein's case:

Theorem 1.3. *Given any graph G, all its r-th roots in $\mathcal{GIRTH}^{+}_{\geq 2r+3}$ can be found in polynomial time.*

Next, we use this result to deal with the general case, i.e. when the roots are allowed to have leaves. It turns out that the same girth bound of $2r+3$ admits a positive result:

Theorem 1.4. *The problem r-TH-POWER-OF-$\mathcal{GIRTH}_{\geq 2r+3}$-GRAPH can be solved in polynomial time.*

Our result proves Conjecture 1.2 (for $r \geq 4$) and is in fact stronger. It also improves the result of [10] for $r = 3, g = 10$. Moreover, our algorithm for this problem is constructive and exhaustive in the sense that it finds "all" r-th roots in $\mathcal{GIRTH}_{\geq 2r+3}$ modulo the non-uniqueness of r-th tree roots of graphs, as explained in Section 4.

These positive results have a hardness counterpart:

Theorem 1.5. *The problem r-TH-POWER-OF-$\mathcal{GIRTH}_{\geq g}$-GRAPH is NP-complete for $g \leq r+1$ when r is odd and $g \leq r+2$ when r is even.*

The paper is structured as follows. First we prove some auxiliary results, useful both in the construction of algorithms and in the hardness result. Section 3 contains the main algorithm from Theorem 1.3, which is then used in Section 4 as a building block of the general recognition algorithm from Theorem 1.4. NP-completeness is proved in Section 5.

2. Auxiliary results

Let us fix some terminology. By $\mathrm{dist}_H(u, v)$ we denote the distance from u to v in H. The *d-neighbourhood* of a vertex u in H is the set of vertices of H which are exactly in distance d from u. The 1-neighbourhood (i.e. the set of vertices adjacent to u) will be denoted $N_H(u)$.

Our setup usually involves a pair of graphs G and H on a common vertex set V such that $G = H^r$. We adopt the notation

$$B_v := \{u \in V : \mathrm{dist}_H(u, v) \leq r\} = N_G(v) \cup \{v\}$$

for $v \in V$ (the letter B stands for "ball" of radius r in H). The lack of explicit reference to r and H in this notation should not lead to confusion. It is important that B_v depend only on G.

Almost all previous work on algorithmic aspects of graph powers [14, 4, 8, 9, 10] makes use of a special gadget, called *tail structure*, which, applied to a vertex u in G, ensures that in any r-th root H of G this vertex has the same, pre-determined neighbourhood. Our main observation is that in fact such a tail structure carries a lot more information about H. It pins down not just $N_H(u)$, but also each d-neighbourhood of u in H for $d = 1, \ldots, r$.

Lemma 2.1. *Let $G = H^r$ and suppose that $\{v_0, v_1, \dots, v_r\} \subset V$ is a set of vertices such that $N_G(v_r) = \{v_{r-1}, \dots, v_1, v_0\}$ and $N_G(v_{i+1}) \subset N_G(v_i)$ for all $i = 0, \dots, r-1$, where the inclusions are strict.* [2]

Then the subgraph of H induced by $\{v_0, v_1, \dots, v_r\}$ is a path $v_0 - v_1 - \dots - v_r$ and the d-neighbourhood of v_0 in H is precisely

$$N_G(v_{r-d}) \setminus N_G(v_{r-d+1}) \cup \{v_d\}$$

for all $d = 1, \dots, r$.

Proof. The subgraph K of H induced by $\{v_0, \dots, v_r\}$ is connected — otherwise $N_G(v_r)$ would contain vertices from outside K. Consider any vertex u of K that has an edge to some vertex w outside K. Clearly, $\mathrm{dist}_K(v_r, u) = r$, since otherwise w would be in $N_G(v_r)$. This means that K is a path from v_r to u and u is the only vertex of that path which has edges to vertices outside K. The condition $N_G(v_{i+1}) \subset N_G(v_i)$ now implies that the vertices of this path are arranged as in the conclusion of the lemma. The second conclusion follows easily. ■

Note that the tail structure itself does not enforce any extra constraints on H other than the d-neighbourhoods of v_0.

In the algorithm for r-TH-POWER-OF-$\mathcal{GIRTH}_{\geq 2r+3}$-GRAPH we will need to solve the following tree root problem with additional restrictions imposed on the d-neighbourhoods of a certain vertex:

Problem. RESTRICTED-r-TH-TREE-ROOT
Instance. A graph G, $r \geq 2$, a vertex $v \in V(G)$ and a partition
$V(G) = \{v\} \cup T^{(1)} \cup \dots \cup T^{(r)} \cup T^{(>r)}$.
Question. Is $G = T^r$ for some tree T such that the
d-neighbourhood of v in T is exactly $T^{(d)}$ for $d = 1, \dots, r$?

Lemma 2.2. *There is a constructive polynomial time algorithm for RESTRICTED-r-TH-TREE-ROOT.*

Proof sketch. The neighbourhood-enforcing gadget from Lemma 2.1 can be attached to the given problem instance in such a way that the original graph has a restricted tree root if and only if the modified graph has any tree root (with no restrictions). Then the algorithms of [6, 2] apply to the modified instance. ■

3. Algorithm for roots in $\mathcal{GIRTH}^{+}_{\geq 2r+3}$

In this section we present the algorithm from Theorem 1.3, that is the polynomial time reconstruction of all r-th roots in $\mathcal{GIRTH}^{+}_{\geq 2r+3}$ of a given graph G. There are two structural properties of graphs $H \in \mathcal{GIRTH}^{+}_{\geq 2r+3}$ that will be used freely throughout the proofs:

(*) Every $x \in V(H)$ is of degree at least 2 and the subgraph of H induced by B_x is a tree. This holds since any cycle in H within B_x would have length at most $2r+1$. We shall depict the ball B_x in H in the tree-like fashion.

[2] This assumption (strictness of inclusions) can be removed at the cost of a more complicated statement, but this generality is not needed here.

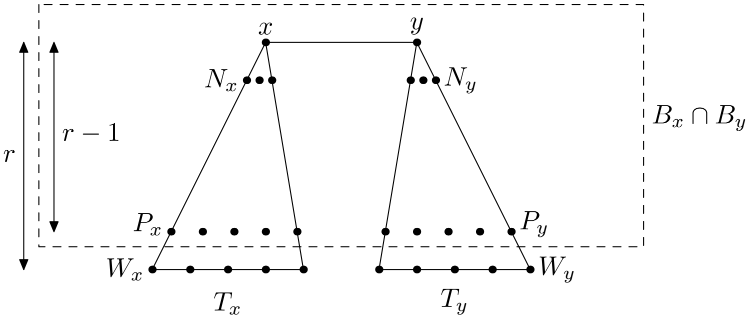


Figure 1: The subgraph of H induced by $B_x \cup B_y$.

(**) If there is a simple path from u to v in H of length exactly $r+1$ or $r+2$ then $u \notin B_v$. Indeed, $u \in B_v$ iff there is a path of length at most r from u to v in H, and combined with the first path this would yield a cycle of length at most $2r+2$.

To describe the algorithm we introduce the following sets:

$$S_{x,y} = B_x \cap B_y \setminus \bigcup_{v \in B_y \setminus B_x} B_v \setminus \{x\}$$

$$P_{x,y} = B_x \cap B_y \cap \bigcup_{v \in S_{x,y}} B_v$$

$$N_{x,y} = B_x \cap B_y \cap \bigcap_{v \in P_{x,y}} B_v \setminus \{x\}$$

Defined for arbitrary $x, y \in V$, these sets are probably quite meaningless for the reader. The definitions are motivated by the proof of the next theorem, in which we determine these sets in more familiar terms for the endpoints x, y of an actual edge in some r-th root of G. Precisely:

Theorem 3.1. *Suppose $G = H^r$ for a graph $H \in \mathcal{GIRTH}^+_{\geq 2r+3}$ and $xy \in E(H)$. Then*

$$N_{x,y} = N_H(x).$$

Proof. Because of the girth condition the set $B_x \cup B_y$ in H consists of two disjoint trees T_x and T_y, rooted in x and y respectively and connected by the edge xy (see Fig.1). Let us introduce some subsets of those trees. By W_x and W_y denote the last levels:

$$W_x = \{u \in T_x : \mathrm{dist}_H(u,x) = r\}, \quad W_y = \{u \in T_y : \mathrm{dist}_H(u,y) = r\},$$

by P_x and P_y the next-to-last levels:

$$P_x = \{u \in T_x : \mathrm{dist}_H(u,x) = r-1\}, \quad P_y = \{u \in T_y : \mathrm{dist}_H(u,y) = r-1\},$$

and by N_x and N_y the children of x and y in T_x and T_y:

$$N_x = \{u \in T_x : \mathrm{dist}_H(u,x) = 1\}, \quad N_y = \{u \in T_y : \mathrm{dist}_H(u,y) = 1\}.$$

Clearly $B_x \cap B_y = (T_x \setminus W_x) \cup (T_y \setminus W_y)$, $W_x = B_x \setminus B_y$ and $W_y = B_y \setminus B_x$. Note that if $r = 2$ we have $N_x = P_x$ and $N_y = P_y$.

First observe that every $u \in N_x$ and every $v \in B_y \setminus B_x = W_y$ are connected by a path of length $r+2$. It follows by (**) that $u \notin B_v$, which implies

$$N_x \subset S_{x,y}.$$

It is also clear that $S_{x,y} \subset T_x$ (because every vertex in T_y has a descendant $v \in W_y$).

Now the sum $\bigcup_{v \in S_{x,y}} B_x \cap B_y \cap B_v$ contains $\bigcup_{v \in N_x} B_x \cap B_y \cap B_v = (B_x \cap B_y) \setminus P_y$. On the other hand, if $v \in S_{x,y}$ and $u \in P_y$ then $u \notin B_v$. Indeed, if $u \in B_v$ then there would be a path from u to v of length at most r. This path cannot be contained in $T_x \cup T_y$ (because $\mathrm{dist}_H(u,x) = r$, so one can only get as far as x going from u), hence it must exit T_y through W_y and then enter T_x through W_x, finally reaching $v \in S_{x,y}$. However, that yields a path from W_y to $S_{x,y}$ of length at most r (in fact at most $r-1$), contradicting the definition of $S_{x,y}$. Eventually we proved

$$P_{x,y} = (B_x \cap B_y) \setminus P_y.$$

Now we have $\{y\} \cup N_x \subset N_{x,y}$ because every vertex of $\{y\} \cup N_x$ is in distance at most r from all the vertices of $(B_x \cap B_y) \setminus P_y$. On the other hand, for every vertex u of $B_x \cap B_y$ that is not in $N_x \cup \{x,y\}$ one can find a path of length $r+1$ that starts in u and ends in a vertex $v \in (B_x \cap B_y) \setminus P_y$. Then, according to (**), $u \notin B_v$, so $u \notin N_{x,y}$. Such a path is obtained by going from u up the tree it is contained in (T_x or T_y) and then down in the other tree.

Concluding, we have identified $N_{x,y}$ to be $N_x \cup \{y\}$, as required. ■

The previous theorem should be understood as follows. Given a graph G, we want to find its r-th root H. If we fix at least one edge xy of H in advance, we can compute the neighbourhood $N_H(x)$ of x using only the data available in G. But then we can move on in the same way, computing the neighbours of those neighbours etc.

Algorithm 1 Input: G,r. **Output:** All r-th roots of G in $\mathcal{GIRTH}^+_{\geq 2r+3}$

```
pick a vertex $x$ with smallest $|B_x|$
for all $y$ in $B_x$ do
  $H$ =reconstructFromOneEdge($G, xy$)
  if $H \in \mathcal{GIRTH}^+_{\geq 2r+3}$ and $H^r = G$ output $H$
end for
reconstructFromOneEdge($G, e$):
$H = (V(G), \{e\})$
for all $u \in V$ set processed[$u$]:=false
while $H$ has an unprocessed vertex $x$ of degree at least 1 do
  $y$ = any neighbour of $x$ in $H$
  $E(H) = E(H) \cup \{xz$ for all $z \in N_{x,y}\}$
  processed[$x$]:=true
end while
return $H$
```

The r-th root algorithm is now straightforward. The procedure *reconstructFromOneEdge* attempts to compute H from G assuming the existence of a given edge e in H. This is repeated for all possible edges from a fixed vertex x. It remains to notice that $N_{x,y}$ can be computed in polynomial time.

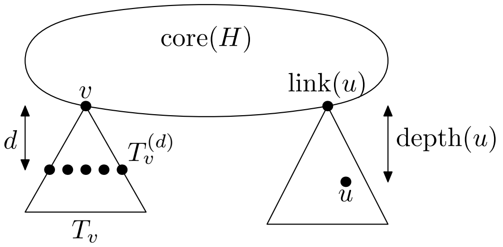


Figure 2: The notation of Section 4.

4. Removing the no-leaves restriction

In this section we obtain a polynomial time algorithm for the general recognition problem r-TH-POWER-OF-$\mathcal{GIRTH}_{\geq 2r+3}$-GRAPH, proving Theorem 1.4. We start with a few definitions (see Fig.2).

For a graph H, which is not a tree, let core(H) denote the largest induced subgraph of H whose every vertex has degree at least two. Alternatively this can be defined as follows. Given H, let H' be the graph obtained from H by removing all *leaves* (vertices of degree one) and inductively define $H^{(1)} = H'$, $H^{(n)} = (H^{(n-1)})'$. This process eventually stabilizes at the graph core(H).

A vertex $v \in V(H)$ is called a *core vertex* if it belongs to core(H) and a *non-core vertex* otherwise. The non-core vertices are grouped into trees attached to the core. For every vertex $v \in$ core(H) we denote by T_v the tree attached at v (including v) and by $T_v^{(d)}$ (for $d \geq 0$) the set of vertices of T_v located in distance d from v. For a non-core vertex u the *link* of u (denoted link(u)) is its closest core vertex and the *depth* of u (denoted depth(u)) is the distance from u to link(u).

4.1. Outline of the algorithm.

The algorithm for r-TH-POWER-OF-$\mathcal{GIRTH}_{\geq 2r+3}$-GRAPH processes the input graph G in several steps (see Algorithm 2). First, we check if G has a tree r-th root [6, 2]. If not, then we split the vertices of G into the core and non-core vertices of any of its r-th roots. Lemma 4.1 shows how to find such a partition and ensures that it is uniquely determined only by the graph G.

Let $\tilde{G}$ be the subgraph of G induced by all the vertices that are classified as belonging to the core of any possible r-th root H. We now employ the algorithm from the previous section to find all r-th roots $\tilde{H}$ of $\tilde{G}$ which have girth at least $2r+3$ and no leaves (there are at most $\delta(G)$ of them; conjecturally there is at most one).

Finally, we must attach the non-core vertices to each of the possible $\tilde{H}$. It turns out that once the core is fixed, the link of each non-core vertex can be uniquely determined, so we can pin down all the sets $V(T_v)$. However, we cannot simply look for any r-th tree root of the subgraph of G induced by $V(T_v)$, because we have to ensure that the tree structure that we are going to impose on $V(T_v)$ is compatible with the neighbourhood information contained in the rest of G. Fortunately Lemma 4.2 guarantees that for a fixed G and core(H), all the sets $T_v^{(d)}$ for $d = 1, \ldots, r$ are also uniquely determined. Since all the distances from the vertices of T_v to the rest of the graph depend only on the vertex depths and the structure of the core, this is exactly the additional piece of data we need. Any tree root satisfying the given depth constraints will be compatible with the rest of the graph. Concluding, the problem we are left with for each T_v is the RESTRICTED-r-TH-TREE-ROOT from Section 2. If all these instances have positive solutions, then the graph H defined as $\tilde{H}$ with the trees T_v attached at each core vertex v is an r-th root of G.

The next two subsections describe the two crucial steps: detecting non-core vertices and the reconstruction of trees T_v.

4.2. Finding core and non-core vertices.

The next lemma shows how to detect all vertices located "close to the bottom" of the trees T_v in H.

Lemma 4.1. *Suppose $H \in \mathcal{GIRTH}_{\geq 2r+3}$ and $H^r = G$.Then the following conditions are equivalent for a vertex $u \in H$:*

(1) *$u \notin H^{(r)}$.*
(2) *There is some vertex $v \in H$, $v \neq u$ such that $B_u \subseteq B_v$.*

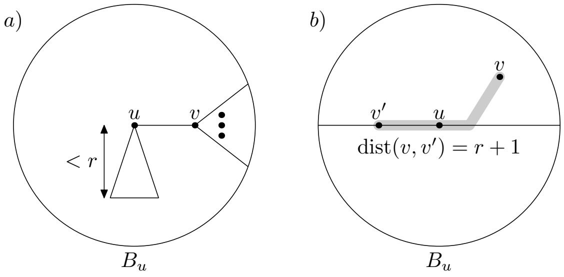


Figure 3: The proof of Lemma 4.1.

Proof. If $u \notin H^{(r)}$, then by the definition u becomes a leaf after at most $r - 1$ steps of the leaf-removal procedure and is removed in the subsequent step. Let v be the last vertex adjacent to u just before u is removed (see Fig.3a). Clearly all the vertices reachable from u in at most r steps are also reachable from v in at most r steps, so $B_u \subseteq B_v$.

If, on the other hand, $u \in H^{(r)}$ then u is not removed in the first r steps of cutting off the leaves of H, which means there exist at least two disjoint paths of length r starting at u (see Fig.3b). However, it implies that for every vertex $v \in B_u$ there exists another $v' \in B_u$ (on one of those paths) such that $\mathrm{dist}_H(v, v') = r + 1$, hence $v' \in B_u \setminus B_v$. Therefore B_u is not contained in B_v for any $v \neq u$. ■

Recursively deleting all vertices u such that $B_u \subseteq B_v$ for some $v \neq u$ determines the consecutive sets $V(H^{(r)})$, $V(H^{(2r)})$, $V(H^{(3r)})$, ... for *any* r-th root $H \in \mathcal{GIRTH}_{\geq 2r+3}$ of G using only the information available in G. Eventually we obtain $V(\mathrm{core}(H))$ which is the vertex set of $\tilde{G}$.

4.3. Attaching the trees T_v.

For each possible core(H) we need to decide on a way of attaching the remaining (non-core) vertices to H in a way which ensures that $H^r = G$. It turns out that all the data necessary to ensure the compatibility can be read off from G and core(H), so again this data is common for all the possible r-th roots of G that have a fixed core.

Lemma 4.2. *Suppose that $H \in \mathcal{GIRTH}_{\geq 2r+3}$ is a graph such that H is not a tree and $H^r = G$. Then for every non-core vertex u of H we have:*

- *either $B_u \cap V(\mathit{core}(H)) = \emptyset$, in which case $\mathit{depth}(u) > r$, or*
- *the subgraph of H induced by $B_u \cap V(\mathit{core}(H))$ is a tree whose only center is $\mathit{link}(u)$ and whose height (the distance from the center to every leaf) is $r - \mathit{depth}(u)$.*

Proof. The first statement is obvious. As for the second, the subgraph induced by $B_u \cap V(\mathrm{core}(H))$ consists of all the vertices of $V(\mathrm{core}(H))$ in distance at most $r - \mathrm{depth}(u)$ from $\mathrm{link}(u)$. Since $\mathrm{core}(H)$ is a graph of girth at least $2r + 3$ with no degree one nodes, these vertices induce a tree in H, and all the leaves of this tree are exactly in distance $r - \mathrm{depth}(u)$ from $\mathrm{link}(u)$. Therefore $\mathrm{link}(u)$ is the unique center of that tree. ■

Lemma 4.2 yields a method of partitioning the non-core vertices into the sets $V(T_v)$ and subdividing each $V(T_v)$ into a disjoint union $\{v\} \cup T_v^{(1)} \cup \ldots \cup T_v^{(r)} \cup T_v^{(>r)}$ of vertices in distance $1, 2, \ldots, r$ and more than r from v using only the data from G and $\mathrm{core}(H)$. Indeed, for the vertices u with $B_u \cap V(\mathrm{core}(H)) \neq \emptyset$ one finds the center and height of the subtree of $\mathrm{core}(H)$ induced by $B_u \cap V(\mathrm{core}(H))$ and applies the second part of Lemma 4.2 to obtain both $\mathrm{link}(u)$ and $\mathrm{depth}(u)$, thus classifying u to the appropriate $T_v^{(d)}$. The links of all remaining vertices are determined using the fact that all vertices in one connected component of $G \setminus \bigcup_{v \in \mathrm{core}(H), d=0,\ldots,r-1} T_v^{(d)}$ have the same link.

Algorithm 2
Input: G,r.
Output: r-th roots of G in $\mathcal{GIRTH}_{\geq 2r+3}$ (one per each core)

```
check if $G = T^r$ for some tree $T$

$\tilde{G} := G$
while $\tilde{G}$ has vertices $u, v$ with $B_u \subseteq B_v$ do
  remove from $\tilde{G}$ all $u$ such that $B_u \subseteq B_v$ for some $v$
end while

for every graph $\tilde{H} \in \mathcal{GIRTH}^{+}_{\geq 2r+3}$ such that $\tilde{H}^r = \tilde{G}$ do
  $H := \tilde{H}$
  for every vertex $v \in V(\tilde{H})$ do
    find $V(T_v)$ and a partition $V(T_v) = \{v\} \cup T_v^{(1)} \cup \ldots \cup T_v^{(r)} \cup T_v^{(>r)}$
    use $restrictedTreeRoot$ to reconstruct some tree $T_v$
    extend $H$ by attaching $T_v$ at $v$
  end for
  if all $T_v$ existed output $H$
end for
```

5. Hardness results

Now we sketch the hardness of recognition for powers of graphs of lower-bounded girth (Theorem 1.5). For the reductions we use the following NP-complete problem (see [5, Prob. SP4]). It has already been successfully applied in this context ([4, 8, 9, 10]).

Problem. HYPERGRAPH 2-COLORABILITY (H2C)
Instance. A finite set S and a collection $S_1, \dots, S_m$ of subsets of S.
Question. Can the elements of S be colored with two colors A, B such that each set S_j has elements of both colors?

An instance of this problem (also known as SET-SPLITTING) will be denoted $\mathcal{S} = (S; S_1, \dots, S_m)$. We shall refer to the elements of the universum S as $x_1, \dots, x_n$. Any assignment of colors A and B to the elements of S which satisfies the requirements of the problem will be called a *2-coloring*.

In this section we fix r and let $k = \lfloor \frac{r}{2} \rfloor$, so that $r = 2k$ or $r = 2k+1$ depending on parity.

5.1. Case of odd $r = 2k + 1$

Consider an instance $\mathcal{S} = (S; S_1, \dots, S_m)$ of H2C. The following two definitions describe an auxiliary graph that will be used as a base for further constructions. The reader is referred to Fig.4 for a self-explanatory presentation of the graphs $K_\mathcal{S}$ and $H_\mathcal{S}$ defined below.

Definition 5.1. For an instance $\mathcal{S} = (S; S_1, \dots, S_m)$ let $V_\mathcal{S}$ be the following set of vertices:

- S_j, x_i for all subsets and elements,
- A, B, X,
- $T_{i,j}^{(l)}$ for every pair i, j such that $x_i \in S_j$ and every $l = 1, \dots, k-1$,
- $P_i^{(l)}$ for every x_i and every $l = 1, \dots, k-1$,
- the tail vertices $S_j^{(l)}$ for each j and $l = 1, \dots, r$.

Definition 5.2. Given any instance $\mathcal{S} = (S; S_1, \dots, S_m)$ define a graph $K_\mathcal{S}$ on the vertex set $V_\mathcal{S}$ with the following edges:

- a path $S_j - T_{i,j}^{(1)} - \dots - T_{i,j}^{(k-1)} - x_i$ whenever $x_i \in S_j$,
- a path $x_i - P_i^{(1)} - \dots - P_i^{(k-1)}$ for every x_i,
- $X - x_i$ for all i,
- the tail paths, that is $S_j - S_j^{(1)} - S_j^{(2)} - \dots - S_j^{(r)}$ for every j.

This graph encodes only the structure of $\mathcal{S}$. To encode the coloring we link the loose paths from x_i to either A or B.

Definition 5.3. Given an instance $\mathcal{S}$ and a color assignment, define the graph $H_\mathcal{S}$ to be $K_\mathcal{S}$ with the additional edges $P_i^{(k-1)} - A$ whenever x_i has color A and $P_i^{(k-1)} - B$ whenever x_i has color B.

Note that $H_\mathcal{S}$ has girth $2k + 2 = r + 1$. Now comes the graph to be used in our NP-completeness reduction:

Definition 5.4. For any instance $\mathcal{S} = (S; S_1, \dots, S_m)$ of H2C put

$$G_\mathcal{S} = {K_\mathcal{S}}^r \cup E_\mathcal{S}$$

where $E_\mathcal{S}$ is the set of edges from A and B to each of X, x_i, S_j, $T_{i,j}^{(l)}$, $P_i^{(l)}$, and $S_j^{(1)}$ for all possible i, j, l.

Observe that $G_\mathcal{S}$ is defined independently of any particular color assignment. Moreover, by analyzing Fig.4 it is not hard to check the following lemma:

Lemma 5.5. *For any 2-colored instance $\mathcal{S}$ we have $G_\mathcal{S} = {H_\mathcal{S}}^r$.* ■

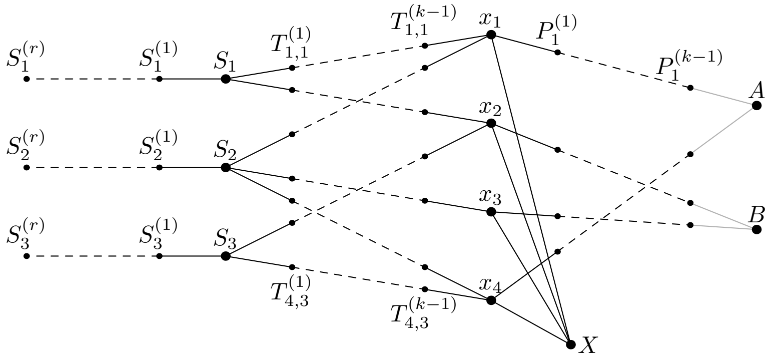


Figure 4: For $\mathcal{S} = (\{x_1, \ldots, x_4\}; \{x_1, x_2\}, \{x_1, x_3, x_4\}, \{x_2, x_4\})$ the graph $K_{\mathcal{S}}$ consists of all *but* the shaded edges. The graph $H_{\mathcal{S}}$ (made of all the edges above) encodes the coloring with x_1, x_4 of color A and x_2, x_3 of color B. It is a 2-coloring of $\mathcal{S}$ since all S_j are in distance $2k$ from A and B.

Proof of Theorem 1.5 for odd r. Given an instance $\mathcal{S} = (S; S_1, \ldots, S_m)$ construct the graph $G_{\mathcal{S}}$. If $\mathcal{S}$ has a 2-coloring, then $G_{\mathcal{S}}$ is the r-th power of a graph with girth at least $r+1$, namely $G_{\mathcal{S}} = {H_{\mathcal{S}}}^r$ by Lemma 5.5.

For the inverse implication suppose that $G_{\mathcal{S}} = H^r$ for some graph H. Define the coloring as follows: x_i has color A (resp. B) if there is a path of length at most k from x_i to A (resp. B) in H. Clearly each x_i is assigned at most one color since otherwise A and B would be adjacent in H^r.

The tail structure $S_j, S_j^{(1)}, \ldots, S_j^{(r)}$ of each S_j satisfies the assumptions of Lemma 2.1, so it enforces that in H:

- for every j the k-neighbourhood of S_j is precisely $\{x_i : x_i \in S_j\} \cup \{S_j^{(k)}\}$ (as in $K_{\mathcal{S}}$),
- A and B are exactly in distance $2k$ from each S_j (by the definition of $E_{\mathcal{S}}$).

Therefore for each j there has to be at least one vertex in $\{x_i : x_i \in S_j\}$ that is k steps from A and at least one that is k steps from B. This proves that the obtained coloring solves the H2C instance. ■

5.2. Case of even $r = 2k$

We omit this case for reasons of space. The argument is similar, but requires a slight modification to the graphs $K_{\mathcal{S}}$, $H_{\mathcal{S}}$ and $G_{\mathcal{S}}$.

6. Conclusions and open problems

In this work we presented an efficient algorithmic solution to Levenshtein's reconstruction conjecture and we applied it to a more general, unrestricted r-th root problem. From a high-level perspective, it was possible because we could extract the "core of the problem" which has very few solutions (as the conjecture suggests), so we could hope that these can be found quickly. We also hope that the reverse flow of ideas is possible, so that some improved algorithmic edge-by-edge reconstruction technique might help resolve Levenshtein's conjecture.

Another (probably challenging) problem is to find a complete girth-parametrized complexity dichotomy, that is to close the gap between $r+1$ (or $r+2$) and $2r+3$. We believe that the r-th power recognition remains NP-complete even for graphs of girth $2r$.

In fact it would even be very interesting to investigate possible complexity results for finding square roots in $\mathcal{GIRTH}_{\geq 5}$ or $\mathcal{GIRTH}^{+}_{\geq 5}$ (completing the complexity dichotomy of [4]). Note that the complete graph $G = K_n$ has a square root in the class $\mathcal{GIRTH}^{+}_{\geq 5}$ if and only if there exists a graph on n vertices that has girth 5 and diameter 2. By the Hoffman-Singleton theorem (see [16, 1]) such a graph may exist only for $n = 5, 10, 50$ and 3250. The first three of these graphs are known, and the existence of the last one (for $n = 3250$) is a long-standing open problem. Therefore, any efficient algorithm for SQUARE-OF-$\mathcal{GIRTH}^{+}_{\geq 5}$-GRAPH might (at least in principle) solve this problem.

Acknowledgement

The authors thank the anonymous STACS referees for helpful comments.

References

[1] N.Biggs, Algebraic Graph Theory, Cambridge Univ. Press

[2] Maw-Shang Chang, Ming-Tat Ko, Hsueh-I Lu, *Linear-Time Algorithms for Tree Root Problems*, Proc. 10th SWAT, LNCS 4059 (2006)

[3] F.Escalante, L.Montejano, T.Rojano, *Characterization of n-path graphs and of graphs having nth root*, Journal of Combinatorial Theory, Series B, 16: 282-298 (1974)

[4] Babak Farzad, Lap Chi Lau, Van Bang Le, Nguyen Ngoc Tuy, *Computing Graph Roots Without Short Cycles*, Proc. 26th STACS (2009) 397-408

[5] M.R.Garey, D.S.Johnson, *Computers and Intractability — A Guide to the Theory of NP-Completeness*, Freeman, Oxford, UK, 1979

[6] P.E.Kearney, D.G.Corneil *Tree powers*, Journal of Algorithms 29 (1998) 111-131

[7] Martin Kutz, *The complexity of Boolean matrix root computation*, Theor. Comp. Sci. 325 (2004) 373-390

[8] Lap Chi Lau, *Bipartite Roots of Graphs*, ACM Transactions on Algorithms, Vol.2, No.2, April 2006, 178-208

[9] Lap Chi Lau, Derek G. Corneil *Recognizing Powers of Proper Interval, Split and Chordal Graphs*, SIAM J. Discrete Math., Vol.18, No.1, 2004, 83-102

[10] Van Bang Le and Ngoc Tuy Nguyen, *Hardness Results and Efficient Algorithms for Graph Powers*, WG 2009

[11] V.I. Levenshtein, *A conjecture on the reconstruction of graphs from metric balls of their vertices*, Discrete Mathematics 308(5-6): 993-998 (2008)

[12] V.I. Levenshtein, E.V. Konstantinova, E.Konstantinov, S.Molodtsov, *Reconstruction of a graph from 2-vicinities of its vertices*, Discrete Applied Mathematics 156(9): 1399-1406 (2008)

[13] Y.-L.Lin, S.S.Skiena, *Algorithms for square roots of graphs*, SIAM J. Discrete Math. 8 (1995), 99-118

[14] R.Motwani, M.Sudan, *Computing Roots of Graphs is Hard*, Discrete Applied Mathematics 54(1): 81-88 (1994)

[15] A.Mukhopadhyay, *The square root of a graph*, Journal of Combinatorial Theory, Series B, 2: 290-295 (1967)

[16] R.R.Singleton, *There is no irregular Moore graph*, American Mathematical Monthly 75, vol 1 (1968) 42-43

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 47-58
www.stacs-conf.org

THE TROPICAL DOUBLE DESCRIPTION METHOD

XAVIER ALLAMIGEON [1] AND STÉPHANE GAUBERT [2] AND ÉRIC GOUBAULT [3]

[1] Direction du Budget, 4ème sous-direction, Bureau des transports, Paris, France

[2] INRIA Saclay and CMAP, Ecole Polytechnique, France

[3] CEA, LIST MeASI – Gif-sur-Yvette, France
E-mail address: `firstname.lastname@{polytechnique.org,inria.fr,cea.fr}`

Abstract. We develop a tropical analogue of the classical double description method allowing one to compute an internal representation (in terms of vertices) of a polyhedron defined externally (by inequalities). The heart of the tropical algorithm is a characterization of the extreme points of a polyhedron in terms of a system of constraints which define it. We show that checking the extremality of a point reduces to checking whether there is only one minimal strongly connected component in an hypergraph. The latter problem can be solved in almost linear time, which allows us to eliminate quickly redundant generators. We report extensive tests (including benchmarks from an application to static analysis) showing that the method outperforms experimentally the previous ones by orders of magnitude. The present tools also lead to worst case bounds which improve the ones provided by previous methods.

Introduction

Tropical polyhedra are the analogues of convex polyhedra in tropical algebra. The latter deals with structures like the max-plus semiring $\mathbb{R}_{\max}$ (also called *max-plus algebra*), which is the set $\mathbb{R} \cup \{-\infty\}$, equipped with the addition $x \oplus y := \max(x, y)$ and the multiplication $x \otimes y := x + y$.

The study of the analogues of convex sets in tropical or max-plus algebra is an active research topic, and has been treated under various guises. It arose in the work of Zimmerman [Zim77], following a way opened by Vorobyev [Vor67], motivated by optimization theory. Max-plus cones were studied by Cuninghame-Green [CG79]. Their theory was independently developed by Litvinov, Maslov and Shpiz [LMS01] (see also [MS92]) with

1998 ACM Subject Classification: F.2.2.Geometrical problems and computations, G.2.2 Hypergraphs; Algorithms, Verification.

Key words and phrases: convexity in tropical algebra, algorithmics and combinatorics of tropical polyhedra, computational geometry, discrete event systems, static analysis.

This work was performed when the first author was with EADS Innovation Works, SE/IA – Suresnes, France and CEA, LIST MeASI – Gif-sur-Yvette, France.

This work was partially supported by the Arpege programme of the French National Agency of Research (ANR), project "ASOPT", number ANR-08-SEGI-005 and by the Digiteo project DIM08 "PASO" number 3389.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2443

motivations from variations calculus and asymptotic analysis, and by Cohen, Gaubert, and Quadrat [CGQ04] who initiated a "geometric approach" of discrete event systems [CGQ99], further developed in [Kat07, DLGKL09]. Other motivations arise from abstract convexity, see the book by Singer [Sin97], and also the work of Briec and Horvath [BH04]. The field has attracted recently more attention after the work of Develin and Sturmfels [DS04], who pointed out connections with tropical geometry, leading to several works by Joswig, Yu, and the same authors [Jos05, DY07, JSY07, Jos09].

A tropical polyhedron can be represented in two different ways, either internally, in terms of extreme points and rays, or externally, in terms of linear inequalities (see Sect. 1 for details). As in the classical case, passing from the external description of a polyhedron to its internal description is a fundamental computational issue. This is the object of the present paper.

Butkovič and Hegedus [BH84] gave an algorithm to compute the generators of a tropical polyhedral cone described by linear inequalities. Gaubert gave a similar one and derived the equivalence between the internal and external representations [Gau92, Ch. III] (see [GK09] for a recent discussion). Both algorithms rely on a successive elimination of inequalities, but have the inconvenience of squaring at each step the number of candidate generators, unless an elimination technique is used, as in the Maxplus toolbox of Scilab [CGMQ]. Joswig developed a different approach, implemented in Polymake [GJ], in which a tropical polytope is represented as a polyhedral complex [DS04, Jos09].

The present work grew out from two applications: to discrete event systems [Kat07, DLGKL09], and to software verification by static analysis [AGG08]. In these applications, passing from the external to the internal representation is a central difficulty. A further motivation originates from mean payoff games [AGG09b]. These motivations are reviewed in Section 2.

Contributions. We develop a new algorithm which computes the extreme elements of tropical polyhedra. It is based on a successive elimination of inequalities, and a result (Th. 4.1) allowing one, given a polyhedron $\mathcal{P}$ and a tropical halfspace $\mathcal{H}$, to construct a list of candidates for the generators of $\mathcal{P} \cap \mathcal{H}$. The key ingredient is a combinatorial characterization of the extreme generators of a polyhedron defined externally (Th. 3.5 and 3.7): we reduce the verification of the extremality of a candidate to the existence of a strongly connected component reachable from any other in a directed hypergraph. We include a complexity analysis and experimental results (Sect. 4), showing that the new algorithm outperforms the earlier ones, allowing us to solve instances which were previously by far inaccessible. Our result also leads to worst case bounds improving the ones of previously known algorithms.

1. Definitions: tropical polyhedra and polyhedral cones

The neutral elements for the addition $\oplus$ and multiplication $\otimes$, *i.e.*, the zero and the unit, will be denoted by $\mathbb{0} := -\infty$ and $\mathbb{1} := 0$, respectively. The tropical analogues of the operations on vectors and matrices are defined naturally. The elements of $\mathbb{R}_{\max}^d$, the dth fold Cartesian product of $\mathbb{R}_{\max}$, will be thought of as vectors, and denoted by bold symbols, like $\boldsymbol{x} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_d)$.

A *tropical halfspace* is a set of the vectors $\boldsymbol{x} = (\boldsymbol{x}_i) \in \mathbb{R}_{\max}^d$ verifying an inequality constraint of the form

$$\max_{1 \leq i \leq d} a_i + \boldsymbol{x}_i \leq \max_{1 \leq i \leq d} b_i + \boldsymbol{x}_i, \quad a_i, b_i \in \mathbb{R}_{\max}.$$

A *tropical polyhedral cone* is defined as the intersection of n halfspaces. It can be equivalently written as the set of the solutions of a system of inequality constraints $A\boldsymbol{x} \leq B\boldsymbol{x}$. Here, $A = (a_{ij})$ and $B = (b_{ij})$ are $n \times d$ matrices with entries in $\mathbb{R}_{\max}$, concatenation denotes the matrix product (with the laws of $\mathbb{R}_{\max}$), and $\leq$ denotes the standard partial ordering of vectors. For sake of readability, tropical polyhedral cones will be simply referred to as *polyhedral cones* or *cones*.

Tropical polyhedral cones are known to be generated by their extreme rays [GK06, GK07, BSS07]. Recall that a *ray* is the set of scalar multiples of a non-zero vector $\boldsymbol{u}$. It is *extreme* in a cone $\mathcal{C}$ if $\boldsymbol{u} \in \mathcal{C}$ and if $\boldsymbol{u} = \boldsymbol{v} \oplus \boldsymbol{w}$ with $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{C}$ implies that $\boldsymbol{u} = \boldsymbol{v}$ or $\boldsymbol{u} = \boldsymbol{w}$. A finite set $G = (\boldsymbol{g}^i)_{i \in I}$ of vectors is said to *generate* a polyhedral cone $\mathcal{C}$ if each $\boldsymbol{g}^i$ belongs to $\mathcal{C}$, and if every vector $\boldsymbol{x}$ of $\mathcal{C}$ can be written as a *tropical linear combination* $\bigoplus_i \lambda_i \boldsymbol{g}^i$ of the vectors of G (with $\lambda_i \in \mathbb{R}_{\max}$). Note that in tropical linear combinations, the requirement that λ_i be nonnegative is omitted. Indeed, $\mathbb{0} = -\infty \leq \lambda$ holds for all scalar $\lambda \in \mathbb{R}_{\max}$.

The tropical analogue of the Minkowski theorem [GK07, BSS07] shows in particular that every generating set of a cone that is minimal for inclusion is obtained by selecting precisely one (non-zero) element in each extreme ray.

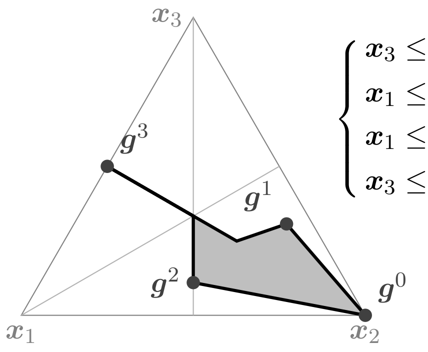
A tropical polyhedron of $\mathbb{R}_{\max}^d$ is the affine analogue of a tropical polyhedral cone. It is defined by a system of inequalities of the form $A\boldsymbol{x} \oplus \boldsymbol{c} \leq B\boldsymbol{x} \oplus \boldsymbol{d}$. It can be also expressed as the set of the tropical affine combinations of its generators. The latter are of the form $\bigoplus_{i \in I} \lambda_i \boldsymbol{v}^i \ \oplus \ \bigoplus_{j \in J} \mu_j \boldsymbol{r}^j$, where the $(\boldsymbol{v}^i)_{i \in I}$ are the extreme points, the $(\boldsymbol{r}^j)_{j \in J}$ a set formed by one element of each extreme ray, and $\bigoplus_i \lambda_i = \mathbb{1}$. It is known [CGQ04, GK07] that every tropical polyhedron of $\mathbb{R}_{\max}^d$ can be represented by a tropical polyhedral cone of $\mathbb{R}_{\max}^{d+1}$ thanks to an analogue of the homogenization method used in the classical case (see [Zie98, Sect. 1.5]). Then, the extreme rays of the cone are in one-to-one correspondence with the extreme generators of the polyhedron. That is why, in the present paper, we will only state the main results for cones, leaving to the reader the derivation of the affine analogues, along the lines of [GK07].

In the sequel, we will illustrate our results on the polyhedral cone $\mathcal{C}$ given in Fig. 1, defined by the system in the right side. The left side is a representation of $\mathcal{C}$ in barycentric coordinates: each element $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$ is represented as a barycenter with weights $(e^{\boldsymbol{x}_1}, e^{\boldsymbol{x}_2}, e^{\boldsymbol{x}_3})$ of the three vertices of the outermost triangle. Then two elements of a same ray are represented by the same point. The cone $\mathcal{C}$ is depicted in solid gray (the black border is included), and is generated by the extreme elements $\boldsymbol{g}^0 = (\mathbb{0}, 0, \mathbb{0})$, $\boldsymbol{g}^1 = (-2, 1, 0)$, $\boldsymbol{g}^2 = (2, 2, 0)$, and $\boldsymbol{g}^3 = (0, \mathbb{0}, 0)$.

2. Motivations from static analysis, discrete event systems, and mean payoff games

Tropical polyhedra have been recently involved in static analysis by abstract interpretation [AGG08]. It has been shown that they allow to automatically compute complex invariants involving the operators min and max which hold over the variables of a program. Such invariants are disjunctive, while most existing techniques in abstract interpretation are only able to express conjunctions of affine constraints, see in particular [CC77, CH78, Min01].

For instance, tropical polyhedra can handle notorious problems in verification of memory manipulations. Consider the well-known memory string manipulating function `memcpy` in C. A call to `memcpy(dst, src, n)` copies exactly the first `n` characters of the string buffer



$$\begin{cases} \boldsymbol{x}_3 \leq \boldsymbol{x}_1 + 2 \\ \boldsymbol{x}_1 \leq \max(\boldsymbol{x}_2, \boldsymbol{x}_3) \\ \boldsymbol{x}_1 \leq \boldsymbol{x}_3 + 2 \\ \boldsymbol{x}_3 \leq \max(\boldsymbol{x}_1, \boldsymbol{x}_2 - 1) \end{cases}$$

Figure 1: A tropical polyhedral cone in $\mathbb{R}^3_{\max}$

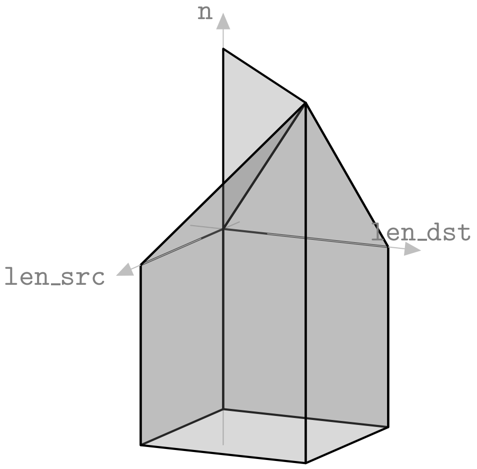


Figure 2: `memcpy` invariant

`src` to `dst`. In program verification, precise invariants over the length of the strings are needed to ensure the absence of string buffer overflows. Recall that the length of a string is defined as the position of the first null character in the string. To precisely analyze the function `memcpy`, two cases have to be distinguished:

(i) either `n` is strictly smaller than the source length `len_src`, so that only non-null characters are copied into `dst`, hence `len_dst` $\geq$ `n`,

(ii) or `n` $\geq$ `len_src` and the null terminal character of `src` will be copied into `dst`, thus `len_dst` $=$ `len_src`.

Thanks to tropical polyhedra, the invariant $\min(\texttt{len\_src}, \texttt{n}) = \min(\texttt{len\_dst}, \texttt{n})$, or equivalently $\max(-\texttt{len\_src}, -\texttt{n}) = \max(-\texttt{len\_dst}, -\texttt{n})$, can be automatically inferred. It is the *exact* encoding of the disjunction of the cases (i) and (ii). The invariant is represented by the non-convex set of $\mathbb{R}^3$ depicted in Figure 2. In the application to static analysis, the performance of the algorithm computing the extreme elements of tropical polyhedra plays a crucial role in the scalability of the analyzer (see [AGG08] for further details).

A second motivation arises from the "geometric approach" of max-plus linear discrete event systems [CGQ99], in which the computation of feedbacks ensuring that the state of the system meets a prescribed constraint (for instance that certain waiting times remain bounded) reduces [Kat07] to computing the greatest fixed point of an order preserving map on the set of tropical polyhedra. Similar computations arise when solving dual observability problems [DLGKL09]. Again, the effective handling of these polyhedra turns out to be the bottleneck.

A third motivation arises from the study of mean payoff combinatorial games. In particular, it is shown in [AGG09b] that checking whether a given initial state of a mean payoff game is winning is equivalent to finding a vector in an associated tropical polyhedral cone (with a prescribed finite coordinate). This polyhedron consists of the super-fixed points of the dynamic programming operator (potentials), which certify that the game is winning.

3. Characterizing extremality from inequality constraints

3.1. Preliminaries on extremality

The following lemma, which is a variation on the proof of Th. 3.1 of [GK07] and on Th. 14 of [BSS07], shows that extremality can be expressed as a minimality property:

Proposition 3.1. *Given a polyhedral cone $\mathcal{C} \subset \mathbb{R}_{\max}^d$, $\boldsymbol{g}$ is extreme if and only if there exists $1 \leq t \leq d$ such that $\boldsymbol{g}$ is a minimal element of the set $\{ \boldsymbol{x} \in \mathcal{C} \mid \boldsymbol{x}_t = \boldsymbol{g}_t \}$, i.e. $\boldsymbol{g} \in \mathcal{C}$ and for each $\boldsymbol{x} \in \mathcal{C}$, $\boldsymbol{x} \leq \boldsymbol{g}$ and $\boldsymbol{x}_t = \boldsymbol{g}_t$ implies $\boldsymbol{x} = \boldsymbol{g}$. In that case, $\boldsymbol{g}$ is said to be* extreme of type t.

In Fig. 3, the light gray area represents the set of the elements $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$ of $\mathbb{R}_{\max}^3$ such that $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) \leq \boldsymbol{g}^2$ implies $\boldsymbol{x}_1 < \boldsymbol{g}_1^2$. It clearly contains the whole cone except $\boldsymbol{g}^2$, which shows that $\boldsymbol{g}^2$ is extreme of type 1.

A *tropical segment* is the set of the tropical linear combinations of two points. Using the fact that a tropical segment joining two points of a polyhedral cone $\mathcal{C}$ yields a continuous path included in $\mathcal{C}$, one can check that $\boldsymbol{g}$ is extreme of type t in $\mathcal{C}$ if and only if there is a neighborhood N of $\boldsymbol{g}$ such that $\boldsymbol{g}$ is minimal in $\{ \boldsymbol{x} \in \mathcal{C} \cap N \mid \boldsymbol{x}_t = \boldsymbol{g}_t \}$. Thus, extremality is a local property.

Finally, the extremality of an element $\boldsymbol{g}$ in a cone $\mathcal{C}$ can be equivalently established by considering the vector formed by its non-$\mathbb{0}$ coordinates. Formally, let $\mathsf{supp}(\boldsymbol{x}) := \{i \mid \boldsymbol{x}_i \neq \mathbb{0}\}$ for any $\boldsymbol{x} \in \mathbb{R}_{\max}^d$. Then $\boldsymbol{g}$ is extreme in $\mathcal{C}$ if and only if it is extreme in $\{\boldsymbol{x} \in \mathcal{C} \mid \mathsf{supp}(\boldsymbol{x}) \subset \mathsf{supp}(\boldsymbol{g})\}$. This allows to assume that $\mathsf{supp}(\boldsymbol{g}) = \{1, \dots, d\}$ without loss of generality.

3.2. Expressing extremality using the tangent cone

For now, the polyhedral cone $\mathcal{C}$ is supposed to be defined by a system $A\boldsymbol{x} \leq B\boldsymbol{x}$ of n inequalities.

Consider an element $\boldsymbol{g}$ of the cone $\mathcal{C}$, which we assume, from the previous discussion, to satisfy $\mathsf{supp}(\boldsymbol{g}) = \{ 1, \dots, d \}$. In this context, the *tangent cone* of $\mathcal{C}$ at $\boldsymbol{g}$ is defined as the tropical polyhedral cone $\mathcal{T}(\boldsymbol{g}, \mathcal{C})$ of $\mathbb{R}_{\max}^d$ given by the system of inequalities

$$\max_{i \in \arg\max(A_k \boldsymbol{g})} \boldsymbol{x}_i \leq \max_{j \in \arg\max(B_k \boldsymbol{g})} \boldsymbol{x}_j \qquad \text{for all } k \text{ such that } A_k \boldsymbol{g} = B_k \boldsymbol{g}, \tag{3.1}$$

where for each row vector $\boldsymbol{c} \in \mathbb{R}_{\max}^{1 \times d}$, $\arg\max(\boldsymbol{c}\boldsymbol{g})$ is defined as the argument of the maximum $\boldsymbol{c}\boldsymbol{g} = \max_{1 \leq i \leq d}(\boldsymbol{c}_i + \boldsymbol{g}_i)$, and where A_k and B_k denote the kth rows of A and B, respectively.

The tangent cone $\mathcal{T}(\boldsymbol{g}, \mathcal{C})$ provides a local description of the cone $\mathcal{C}$ around $\boldsymbol{g}$:

Proposition 3.2. *There exists a neighborhood N of $\boldsymbol{g}$ such that for all $\boldsymbol{x} \in N$, $\boldsymbol{x}$ belongs to $\mathcal{C}$ if and only if it is an element of $\boldsymbol{g} + \mathcal{T}(\boldsymbol{g}, \mathcal{C})$.*

As an illustration, Fig. 4 depicts the set $\boldsymbol{g}^2 + \mathcal{T}(\boldsymbol{g}^2, \mathcal{C})$ (in semi-transparent light gray) when $\mathcal{C}$ is the cone given in Fig. 1. Both clearly coincide in the neighborhood of $\boldsymbol{g}^2$.

Since extremality is a local property, it can be equivalently characterized in terms of the tangent cone. Let $\mathbf{1}$ be the element of $\mathbb{R}_{\max}^d$ whose all coordinates are equal to $\mathbb{1}$.

Proposition 3.3. *The element $\boldsymbol{g}$ is extreme in $\mathcal{C}$ iff the vector $\mathbf{1}$ is extreme in $\mathcal{T}(\boldsymbol{g}, \mathcal{C})$.*

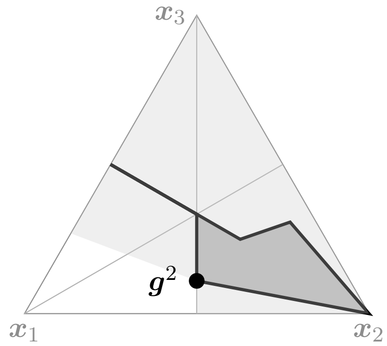


Figure 3: Extremality of $\boldsymbol{g}^2$

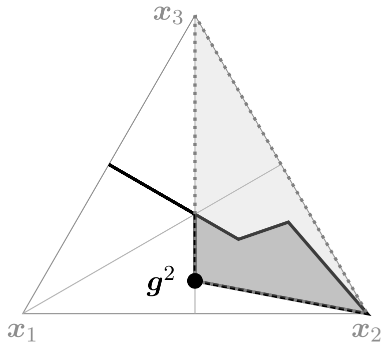


Figure 4: The set $\boldsymbol{g}^2 + \mathcal{T}(\boldsymbol{g}^2, \mathcal{C})$

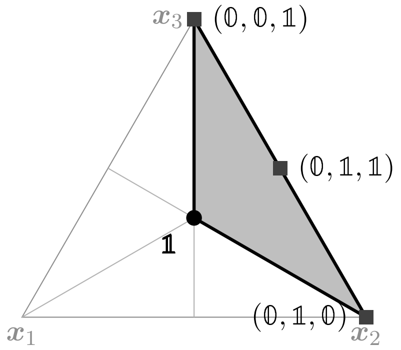

x_3
$(\mathbb{0}, \mathbb{0}, \mathbb{1})$
$(\mathbb{0}, \mathbb{1}, \mathbb{1})$
$\mathbb{1}$
$(\mathbb{0}, \mathbb{1}, \mathbb{0})$
x_1
x_2

Figure 5: The $\{ \mathbb{0}, \mathbb{1} \}$-elements of $\mathcal{T}(\boldsymbol{g}^2, \mathcal{C})$

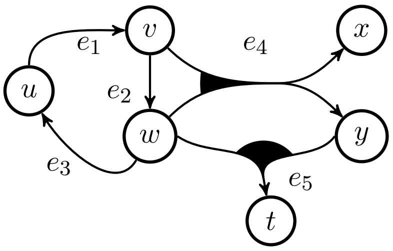


Figure 6: A directed hypergraph

The problem is now reduced to the characterization of the extremality of the vector $\mathbb{1}$ in a $\{ \mathbb{0}, \mathbb{1} \}$*-cone*, *i.e.* a polyhedral cone defined by a system of the form $C\boldsymbol{x} \leq D\boldsymbol{x}$ where $C, D \in \{ \mathbb{0}, \mathbb{1} \}^{n \times d}$. The following proposition states that only $\{ \mathbb{0}, \mathbb{1} \}$-vectors, *i.e.* elements of the tropical regular cube $\{ \mathbb{0}, \mathbb{1} \}^d$, have to be considered:

Proposition 3.4. *Let $\mathcal{D} \subset \mathbb{R}^d_{\max}$ be a $\{ \mathbb{0}, \mathbb{1} \}$-cone. Then $\mathbb{1}$ is extreme of type t if and only if it is the unique element $\boldsymbol{x}$ of $\mathcal{D} \cap \{ \mathbb{0}, \mathbb{1} \}^d$ satisfying $\boldsymbol{x}_t = \mathbb{1}$.*

The following criterion of extremality is a direct consequence of Prop. 3.3 and 3.4:

Theorem 3.5. *Let $\mathcal{C} \subset \mathbb{R}^d_{\max}$ be a polyhedral cone. Then $\boldsymbol{g} \in \mathcal{C}$ is extreme of type t if and only if the vector $\mathbb{1}$ is the unique $\{ \mathbb{0}, \mathbb{1} \}$-element of the tangent cone $\mathcal{T}(\boldsymbol{g}, \mathcal{C})$ whose t-th coordinate is $\mathbb{1}$.*

Figure 5 shows that in our running example, the $\{ \mathbb{0}, \mathbb{1} \}$-elements of $\mathcal{T}(\boldsymbol{g}^2, \mathcal{C})$ distinct from $\mathbb{1}$ (in squares) all satisfy $\boldsymbol{x}_1 = \mathbb{0}$. Naturally, testing, by exploration, whether the set of 2^{d-1} $\{ \mathbb{0}, \mathbb{1} \}$-elements $\boldsymbol{x}$ verifying $\boldsymbol{x}_t = \mathbb{1}$ belonging to $\mathcal{T}(\boldsymbol{g}, \mathcal{C})$ consists only of $\mathbb{1}$ does not have an acceptable complexity. Instead, the approach of the next section will rely on the equivalent formulation of the criterion of Th. 3.5:

$$\forall l \in \{ 1, \dots, d \}, \ \left[\forall \boldsymbol{x} \in \mathcal{T}(\boldsymbol{g}, \mathcal{C}) \cap \{ \mathbb{0}, \mathbb{1} \}^d, \ \boldsymbol{x}_l = \mathbb{0} \implies \boldsymbol{x}_t = \mathbb{0} \right]. \tag{3.2}$$

3.3. Characterizing extremality with directed hypergraphs

A *directed hypergraph* is a couple (N, E) such that each element of E is of the form (T, H) with $T, H \subset N$.

The elements of N and E are respectively called *nodes* and *hyperedges.* Given a hyperedge $e = (T, H) \in E$, the sets T and H represent the *tail* and the *head* of e respectively, and are also denoted by $T(e)$ and $H(e)$. Figure 6 depicts an example of hypergraph whose nodes are u, v, w, x, y, t, and of hyperedges $e_1 = (\{u\}, \{v\})$, $e_2 = (\{v\}, \{w\})$, $e_3 = (\{w\}, \{u\})$, $e_4 = (\{v, w\}, \{x, y\})$, and $e_5 = (\{w, y\}, \{t\})$.

Reachability is extended from digraphs to directed hypergraphs by the following recursive definition: given $u, v \in N$, then v is *reachable from u in $\mathcal{H}$*, which is denoted $u \rightsquigarrow_{\mathcal{H}} v$, if one of the two conditions holds: $u = v$, or there exists $e \in E$ such that $v \in H(e)$ and all the elements of $T(e)$ are reachable from u. In our example, t is reachable from u.

The size $\mathsf{size}(\mathcal{H})$ of a hypergraph $\mathcal{H} = (N, E)$ is defined as $|N| + \sum_{e \in E}(|T(e)| + |H(e)|)$. In the rest of the paper, directed hypergraphs will be simply referred to as hypergraphs.

We associate to the tangent cone $\mathcal{T}(\boldsymbol{g}, \mathcal{C})$ the hypergraph $\mathcal{H}(\boldsymbol{g}, \mathcal{C}) = (N, E)$ defined by:

$$N = \{\, 1, \dots, d \,\} \qquad E = \{\, (\arg\max(B_k \boldsymbol{g}), \arg\max(A_k \boldsymbol{g})) \mid A_k \boldsymbol{g} = B_k \boldsymbol{g},\ 1 \leq k \leq n \,\}\,.$$

The extremality criterion of Eq. (3.2) suggests to evaluate, given an element of $\mathcal{T}(\boldsymbol{g}, \mathcal{C}) \cap \{\, \mathbb{0}, \mathbb{1} \,\}^d$, the effect of setting its l-th coordinate to the other coordinates. Suppose that it has been discovered that $\boldsymbol{x}_l = \mathbb{0}$ implies $\boldsymbol{x}_{j_1} = \dots = \boldsymbol{x}_{j_n} = \mathbb{0}$. For any hyperedge e of $\mathcal{H}(\boldsymbol{g}, \mathcal{C})$ such that $T(e) \subset \{\, l, j_1, \dots, j_n \,\}$, $\boldsymbol{x}$ satisfies: $\max_{i \in H(e)} \boldsymbol{x}_i \leq \max_{j \in T(e)} \boldsymbol{x}_j = \mathbb{0}$, so that $\boldsymbol{x}_i = \mathbb{0}$ for all $i \in H(e)$. Thus, the propagation of the value $\mathbb{0}$ from the l-th coordinate to other coordinates mimicks the inductive definition of the reachability relation from the node l in $\mathcal{H}(\boldsymbol{g}, \mathcal{C})$:

Proposition 3.6. *For all $l \in \{\, 1, \dots, d \,\}$, the statement given between brackets in Eq.* (3.2) *holds if and only if t is reachable from l in the hypergraph $\mathcal{H}(\boldsymbol{g}, \mathcal{C})$.*

Hence, the extremality criterion can be restated thanks to some considerations on the strongly connected components of $\mathcal{H}(\boldsymbol{g}, \mathcal{C})$. The *strongly connected components* (Sccs for short) of a hypergraph $\mathcal{H}$ are the equivalence classes of the equivalence relation $\equiv_{\mathcal{H}}$, defined by $u \equiv_{\mathcal{H}} v$ if $u \rightsquigarrow_{\mathcal{H}} v$ and $v \rightsquigarrow_{\mathcal{H}} u$. They form a partition of the set of nodes of $\mathcal{H}$. They can be partially ordered by the relation $\preceq_{\mathcal{H}}$, defined by $C_1 \preceq_{\mathcal{H}} C_2$ if C_1 and C_2 admit a representative u and v respectively such that $v \rightsquigarrow_{\mathcal{H}} u$ (*beware of the order of v and u in $v \rightsquigarrow_{\mathcal{H}} u$*). Then Prop. 3.6 and Th. 3.5 imply the following statement:

Theorem 3.7. *Let $\mathcal{C} \subset \mathbb{R}^d_{\max}$ be a polyhedral cone, and $\boldsymbol{g} \in \mathcal{C}$. Then $\boldsymbol{g}$ is extreme if and only if the set of the* Sccs *of the hypergraph $\mathcal{H}(\boldsymbol{g}, \mathcal{C})$, partially ordered by $\preceq_{\mathcal{H}(\boldsymbol{g}, \mathcal{C})}$, admits a least element.*

This theorem is reminiscent of a classical result, showing that a point of a polyhedron defined by inequalities is extreme if and only if the family of gradients of active inequalities at this point is of full rank. Here, the hypergraph encodes precisely the subdifferentials (set of generalized gradients) of the active inequalities but a major difference is that the rank condition must be replaced by the above minimality condition, which is essentially stronger. Indeed, using this theorem, it is shown in [AGK09] that an important class of tropical polyhedra has fewer extreme rays than its classical analogue.

An algorithm due to Gallo *et al.* [GLPN93] shows that one can compute the set of nodes that are reachable from a given node in linear time in an hypergraph. The following result shows that one can in fact compute the minimal Sccs with almost the same complexity. The algorithm is included in the extended version of the present paper [AGG09c]. Although it shows some analogy with the classical Tarjan algorithm, the hypergraph case differs

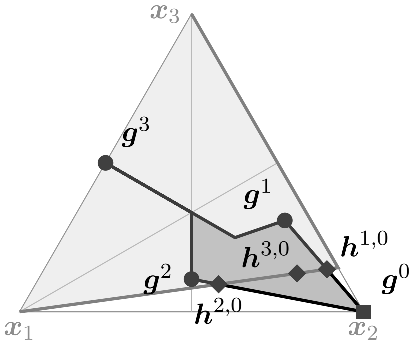


Figure 7: Intersecting a cone with a halfspace

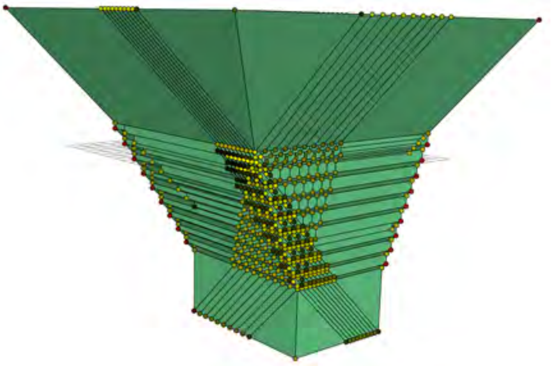


Figure 8: Intersecting 10 affine hyperplanes in dimension 3

critically from the graph case in that one cannot compute all the Sccs using the same technique.

Theorem 3.8. *The set of minimal* Scc*s of a hypergraph* $\mathcal{H} = (N, E)$ *can be computed in time* $O(\mathsf{size}(\mathcal{H}) \times \alpha(|N|))$*, where* α *denotes the inverse of the Ackermann function.*

4. The tropical double description method

Our algorithm is based on a successive elimination of inequalities. Given a polyhedral cone $\mathcal{C}$ defined by a system of n constraints, the algorithm computes by induction on k ($0 \leq k \leq n$) a generating set G_k of the intermediate cone defined by the first k constraints. Then G_n forms a generating set of the cone $\mathcal{C}$. Passing from the set G_k to the set G_{k+1} relies on a result which, given a polyhedral cone $\mathcal{K}$ and a tropical halfspace $\mathcal{H} = \{ \boldsymbol{x} \mid \boldsymbol{ax} \leq \boldsymbol{bx} \}$, allows to build a generating set G' of $\mathcal{K} \cap \mathcal{H}$ from a generating set G of $\mathcal{K}$:

Theorem 4.1. *Let* $\mathcal{K}$ *be a polyhedral cone generated by a set* $G \subset \mathbb{R}^d_{\max}$*, and* $\mathcal{H} = \{ \boldsymbol{x} \mid \boldsymbol{ax} \leq \boldsymbol{bx} \}$ *a tropical halfspace* ($\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^{1\times d}_{\max}$)*. Then the polyhedral cone* $\mathcal{K} \cap \mathcal{H}$ *is generated by the set* $\{ \boldsymbol{g} \in G \mid \boldsymbol{ag} \leq \boldsymbol{bg} \} \ \cup \ \{ (\boldsymbol{ah})\boldsymbol{g} \oplus (\boldsymbol{bg})\boldsymbol{h} \mid \boldsymbol{g}, \boldsymbol{h} \in G,\ \boldsymbol{ag} \leq \boldsymbol{bg},\ and\ \boldsymbol{ah} > \boldsymbol{bh} \}$.

For instance, consider the cone defined in Fig. 1 and the constraint $\boldsymbol{x}_2 \leq \boldsymbol{x}_3 + 2.5$ (depicted in semi-transparent gray in Fig. 7). The three generators $\boldsymbol{g}^1$, $\boldsymbol{g}^2$, and $\boldsymbol{g}^3$ satisfy the constraint, while $\boldsymbol{g}^0$ does not. Their combinations are the elements $\boldsymbol{h}^{1,0}$, $\boldsymbol{h}^{2,0}$, and $\boldsymbol{h}^{3,0}$ respectively. The resulting algorithm is given in Figure 9. As in the classical case, this inductive approach produces redundant generators, hence, the heart of the algorithm is the extremality test in Line 10. We use here the hypergraph characterization (Theorems 3.7 and 3.8).

Complexity analysis. The complexity of the elementary step of ComputeExtreme, *i.e.* the computation of the elements provided by Th. 4.1 and the elimination of non-extreme ones (Lines 7 to 13), can be precisely characterized to $O(nd\alpha(d)\,|G|^2)$, where G is the generating set of the last intermediate cone. By comparison, for classical polyhedra, the same step in the refined double description method by Fukuda and Prodon [FP96] takes a time $O(n\,|G|^3)$. Note that $|G|$ can take values much larger that d.

```
procedure ComputeExtreme($A, B, n$)    ▷ $A, B \in \mathbb{R}_{\max}^{n\times d}$
  if $n = 0$ then    ▷ Base case
    return the tropical canonical basis $(\boldsymbol{\epsilon}_i)_{1\leq i\leq d}$
  else    ▷ Inductive case
    split $A\boldsymbol{x} \leq B\boldsymbol{x}$ into $C\boldsymbol{x} \leq D\boldsymbol{x}$ and $\boldsymbol{a}\boldsymbol{x} \leq \boldsymbol{b}\boldsymbol{x}$, with $C, D \in \mathbb{R}_{\max}^{(n-1)\times d}$ and $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}_{\max}^{1\times d}$
    $G :=$ ComputeExtreme($C, D, n-1$)
    $G^{\leq} := \{\, \boldsymbol{g}^i \in G \mid \boldsymbol{a}\boldsymbol{g}^i \leq \boldsymbol{b}\boldsymbol{g}^i \,\}$, $G^{>} := \{\, \boldsymbol{g}^j \in G \mid \boldsymbol{a}\boldsymbol{g}^j > \boldsymbol{b}\boldsymbol{g}^j \,\}$, $H := G^{\leq}$
    for all $\boldsymbol{g}^i \in G^{\leq}$ and $\boldsymbol{g}^j \in G^{>}$ do
      $\boldsymbol{h} := (\boldsymbol{a}\boldsymbol{g}^j)\boldsymbol{g}^i \oplus (\boldsymbol{b}\boldsymbol{g}^i)\boldsymbol{g}^j$
      if $\boldsymbol{h}$ is extreme in $\{\, \boldsymbol{x} \mid A\boldsymbol{x} \leq B\boldsymbol{x} \,\}$ then
        append $\kappa\boldsymbol{h}$ to $H$, where $\kappa$ is the opposite of the first non-$\mathbb{0}$ coefficient of $\boldsymbol{h}$
      end
    done
  end
  return $H$
end
```

Figure 9: Our main algorithm computing the extreme rays of tropical cones

The overall complexity of the algorithm ComputeExtreme depends on the size of the sets returned in the intermediate steps. In classical geometry, the upper bound theorem of McMullen [McM70] shows that the maximal number of extreme points of a convex polytope in $\mathbb{R}^d$ defined by n inequality constraints is equal to

$$U(n,d) := \binom{n - \lfloor (d+1)/2 \rfloor}{n-d} + \binom{n - \lfloor (d+2)/2 \rfloor}{n-d} \, .$$

The polars of the *cyclic polytopes* (see [Zie98]) are known to reach this bound. Allamigeon, Gaubert, and Katz [AGK09] showed that a similar bound is valid in the tropical setting.

Theorem 4.2 ([AGK09])**.** *The number of extreme rays of a tropical cone in* $(\mathbb{R} \cup \{-\infty\})^d$ *defined as the intersection of* n *tropical half-spaces cannot exceed* $U(n+d, d-1) = O((n+d)^{\lfloor (d-1)/2 \rfloor}$*.*

The bound is asymptotically tight for a fixed n, as d tends to infinity, being approached by a tropical generalization of the (polar of) the cyclic polytope [AGK09]. The bound is believed not to be tight for a fixed d, as n tends to infinity. Finding the optimal bound is an open problem. By combining Theorem 4.2, Theorem 3.8, and Theorem 3.7, we readily get the following complexity result, showing that the execution time is smaller in the tropical case than in the classical case, even with the refinements of [FP96].

Proposition 4.3. *The hypergraph implementation of the tropical double description method returns the set of extreme rays of a polyhedral cone defined by* n *inequalities in dimension* d *in time* $O(n^2 d\alpha(d) G_{\max}^2)$*, where* $G_{\max}$ *is the maximal number of extreme rays of a cone defined by a subsystem of inequalities taken from* $A\boldsymbol{x} \leq B\boldsymbol{x}$*. In particular, the time can be bounded by* $O(n^2 d\alpha(d)(n+d)^{d-1})$*.*

Alternative approaches. The existing approachs discussed in the introduction have a structure which is similar to ComputeExtreme. However, their implementation in the Maxplus toolbox of Scilab [CGMQ] and in our previous work [AGG08] relies on a much less efficient elimination of redundant generators. Its principle is the following: an element $\boldsymbol{h}$ is extreme in the cone generated by a given set H if and only if $\boldsymbol{h}$ can not be expressed as the tropical linear combination of the elements of H which are not proportional to it. This property can

Table 1: Benchmarks on a single core of a 3 GHz Intel Xeon with 3 Gb RAM

	d	n	# final	# inter.	T (s)	T' (s)	T/T'
rnd100	12	15	32	59	0.24	6.72	0.035
rnd100	15	10	555	292	2.87	321.78	$8.9 \cdot 10^{-3}$
rnd100	15	18	152	211	6.26	899.21	$7.0 \cdot 10^{-3}$
rnd30	17	10	1484	627	15.2	4667.9	$3.3 \cdot 10^{-3}$
rnd10	20	8	5153	1273	49.8	50941.9	$9.7 \cdot 10^{-4}$
rnd10	25	5	3999	808	9.9	12177.0	$8.1 \cdot 10^{-4}$
rnd10	25	10	32699	6670	3015.7	—	—
cyclic	10	20	3296	887	25.8	4957.1	$5.2 \cdot 10^{-3}$
cyclic	15	7	2640	740	8.1	1672.2	$5.2 \cdot 10^{-3}$
cyclic	17	8	4895	1589	44.8	25861.1	$1.7 \cdot 10^{-3}$
cyclic	20	8	28028	5101	690	45 days	$1.8 \cdot 10^{-4}$
cyclic	25	5	25025	1983	62.6	8 days	$9.1 \cdot 10^{-5}$
cyclic	30	5	61880	3804	261	—	—
cyclic	35	5	155040	7695	1232.6	—	—

	# var	# lines	T (s)	T' (s)	T/T'
oddeven8	17	118	7.6	152.1	0.050
oddeven9	19	214	128.0	22101.2	$5.8 \cdot 10^{-3}$
oddeven10	21	240	1049.0	—	—

be checked in $O(d \times |H|)$ time using residuation (see [BSS07] for algorithmic details). In the context of our algorithm, the worst case complexity of the redundandy test is therefore $O(d\,|G|^2)$, where G is the set of the extreme rays of the last intermediary cone. This is much worse that our method in $O(nd\alpha(d))$ based on directed hypergraphs, since the cardinality of the set G may be exponential in d (see Theorem 4.2). This is also confirmed by our experiments (see below).

We next sketch a different method relying on arrangement of tropical hyperplanes (arrangements of classical hyperplanes yield naive bounds). Indeed, Theorem 3.7 implies that every extreme ray belongs to the intersection of $d-1$ tropical hyperplanes, obtained by saturating $d-1$ inequalities among the $n+d$ taken from $A\boldsymbol{x} \leq B\boldsymbol{x}$ and $\boldsymbol{x}_i \geq -\infty$, for $i \in [d]$. The max-plus Cramer theorem (see [AGG09a] and the references therein) implies that for generic values of the matrices A, B, every choice of $d-1$ saturated inequalities yields at most one candidate to be an extreme ray, which can be computed in $O(d^3)$ time. This yields a list of $O((n+d)^{d-1})$ candidates, from which the extreme rays can be extracted by using the present hypergraph characterization (Theorems 3.7 and 3.8), leading to a $O((nd\alpha(d) + d^3)(n+d)^{d-1})$ execution time, which is better than the one of Proposition 4.3 by a factor $n/\alpha(d)$ when $n \gg d$. However, the resulting algorithm is of little practical use, since the worst case execution time is essentially always achieved, whereas the double description method takes advantage of the fact that $G_{\max}$ is in general much smaller than the upper bound of Theorem 4.2 (which is probably not optimal in the case $n \gg d$).

A third approach, along the lines of [DS04, Jos09], would consist in representing tropical polyhedra by polyhedral complexes in the usual sense. However, an inconvenient of polyhedral complexes is that their number of vertices (called "pseudo-vertices" to avoid ambiguities) is exponential in the number of extreme rays [DS04]. Hence, the representations used here are more concise. This is illustrated in Figure 8 (generated using Polymake), which shows an intersection of 10 signed tropical hyperplanes, corresponding to the "natural" pattern studied in [AGK09]. There are only 24 extreme rays, but 1215 pseudo-vertices.

Experiments. Allamigeon has implemented Algorithm ComputeExtreme in OCaml, as part of the "Tropical polyhedral library" (`TPLib`), http://penjili.org/tplib.html. Table 1 reports some experiments for different classes of tropical cones: samples formed by several cones chosen randomly (referred to as rnd$x$ where x is the size of the sample), and signed cyclic cones which are known to have a very large number of extreme elements [AGK09].

The successive columns respectively report the dimension d, the number of constraints n, the size of the final set of extreme rays, the mean size of the intermediary sets, and the execution time T (for samples of "random" cones, we give average results).

The result provided by ComputeExtreme does not depend on the order of the inequalities in the initial system. This order may impact the size of the intermediary sets and subsequently the execution time. In our experiments, inequalities are dynamically ordered during the execution: at each step of the induction, the inequality $\boldsymbol{ax} \leq \boldsymbol{bx}$ is chosen so as to minimize the number of combinations $(\boldsymbol{ag}^j)\boldsymbol{g}^i \oplus (\boldsymbol{bg}^i)\boldsymbol{g}^j$. This strategy reports better results than without ordering.

We compare our algorithm with a variant using the alternative extremality criterion which is discussed in Sect. 4 and used in the other existing implementations [CGMQ, AGG08]. Its execution time T' is given in the seventh column. The ratio T/T' shows that our algorithm brings a huge breakthrough in terms of execution time. When the number of extreme rays is of order of 10^4, the second algorithm needs several days to terminate. Therefore, the comparison could not be made in practice for some cases.

Table 1 also reports some benchmarks from applications to static analysis. The experiments oddeven$i$ correspond to the static analysis of the odd-even sorting algorithm of i elements. It is a sort of worst case for our analysis. The number of variables and lines in each program is given in the first columns. The analyzer automatically shows that the sorting program returns an array in which the last (resp. first) element is the maximum (minimum) of the array given as input. It clearly benefits from the improvements of ComputeExtreme, as shown by the ratio with the execution time T' of the previous implementation of the static analyzer [AGG08].

References

[AGG08] X. Allamigeon, S. Gaubert, and É. Goubault. Inferring min and max invariants using max-plus polyhedra. In *SAS'08*, volume 5079 of *LNCS*, pages 189–204. Springer, Valencia, Spain, 2008.

[AGG09a] M. Akian, S. Gaubert, and A. Guterman. Linear independence over tropical semirings and beyond. In G.L. Litvinov and S.N. Sergeev, editors, *Proc. of the International Conference on Tropical and Idempotent Mathematics*, volume 495 of *Contemp. Math.*, pages 1–38. AMS, 2009.

[AGG09b] M. Akian, S. Gaubert, and A. Guterman. Tropical polyhedra are equivalent to mean payoff games. arXiv:0912.2462, 2009.

[AGG09c] X. Allamigeon, S. Gaubert, and E. Goubault. Computing the extreme points of tropical polyhedra. Eprint arXiv:math/0904.3436v2, 2009.

[AGK09] X. Allamigeon, S. Gaubert, and R. D. Katz. The number of extreme points of tropical polyhedra. Eprint arXiv:math/0906.3492, accepted for publication in JCTA, 2009.

[BH84] P. Butkovič and G. Hegedüs. An elimination method for finding all solutions of the system of linear equations over an extremal algebra. *Ekonomicko-matematicky Obzor*, 20, 1984.

[BH04] W. Briec and C. Horvath. $\mathbb{B}$-convexity. *Optimization*, 53:103–127, 2004.

[BSS07] P. Butkovič, H. Schneider, and S. Sergeev. Generators, extremals and bases of max cones. *Linear Algebra Appl.*, 421(2-3):394–406, 2007.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[CG79] R. A. Cuninghame-Green. *Minimax algebra*, volume 166 of *Lecture Notes in Economics and Mathematical Systems*. Springer, 1979.

[CGMQ] G. Cohen, S. Gaubert, M. McGettrick, and J.-P. Quadrat. Maxplus toolbox of scilab. Available at http://minimal.inria.fr/gaubert/maxplustoolbox/; now integrated in ScicosLab. http://www.scicoslab.org.

[CGQ99] G. Cohen, S. Gaubert, and J.P. Quadrat. Max-plus algebra and system theory: where we are and where to go now. *Annual Reviews in Control*, 23:207–219, 1999.

[CGQ04] G. Cohen, S. Gaubert, and J. P. Quadrat. Duality and separation theorem in idempotent semimodules. *Linear Algebra and Appl.*, 379:395–422, 2004.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press.

[DLGKL09] M. Di Loreto, S. Gaubert, R. D. Katz, and J.-J. Loiseau. Duality between invariant spaces for max-plus linear discrete event systems. Eprint arXiv:0901.2915., 2009.

[DS04] M. Develin and B. Sturmfels. Tropical convexity. *Doc. Math.*, 9:1–27 (electronic), 2004.

[DY07] M. Develin and J. Yu. Tropical polytopes and cellular resolutions. *Experimental Mathematics*, 16(3):277–292, 2007.

[FP96] K. Fukuda and A. Prodon. Double description method revisited. In *Selected papers from the 8th Franco-Japanese and 4th Franco-Chinese Conference on Combinatorics and Computer Science*, pages 91–111, London, UK, 1996. Springer.

[Gau92] S. Gaubert. *Théorie des systèmes linéaires dans les dioïdes.* Thèse, École des Mines de Paris, July 1992.

[GJ] E. Gawrilow and M. Joswig. POLYMAKE. http://www.math.tu-berlin.de/polymake/.

[GK06] S. Gaubert and R. Katz. Max-plus convex geometry. In R. A. Schmidt, editor, *RelMiCS/AKA 2006*, volume 4136 of *Lecture Notes in Comput. Sci.*, pages 192–206. Springer, 2006.

[GK07] S. Gaubert and R. Katz. The Minkowski theorem for max-plus convex sets. *Linear Algebra and Appl.*, 421:356–369, 2007.

[GK09] S. Gaubert and R. Katz. Minimal half-spaces and external representation of tropical polyhedra. Eprint arXiv:math/0908.1586, 2009.

[GLPN93] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen. Directed hypergraphs and applications. *Discrete Appl. Math.*, 42(2-3):177–201, 1993.

[Jos05] M. Joswig. Tropical halfspaces. In *Combinatorial and computational geometry*, volume 52 of *Math. Sci. Res. Inst. Publ.*, pages 409–431. Cambridge Univ. Press, Cambridge, 2005.

[Jos09] M. Joswig. Tropical convex hull computations. In G.L. Litvinov and S.N. Sergeev, editors, *Proc. of the International Conference on Tropical and Idempotent Mathematics*, volume 495 of *Contemp. Math.* AMS, 2009.

[JSY07] M. Joswig, B. Sturmfels, and J. Yu. Affine buildings and tropical convexity. *Albanian J. Math.*, 1(4):187–211, 2007.

[Kat07] R. D. Katz. Max-plus (A, B)-invariant spaces and control of timed discrete event systems. *IEEE Trans. Aut. Control*, 52(2):229–241, 2007.

[LMS01] G.L. Litvinov, V.P. Maslov, and G.B. Shpiz. Idempotent functional analysis: an algebraic approach. *Math. Notes*, 69(5):696–729, 2001.

[McM70] P. McMullen. The maximum numbers of faces of a convex polytope. *Mathematika*, 17:179–184, 1970.

[Min01] A. Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, October 2001. `http://www.di.ens.fr/~mine/publi/article-mine-ast01.pdf`.

[MS92] V. Maslov and S. Samborskiĭ, editors. *Idempotent analysis*, volume 13 of *Adv. in Sov. Math.* AMS, RI, 1992.

[Sin97] I. Singer. *Abstract convex analysis.* Wiley, 1997.

[Vor67] N.N. Vorobyev. Extremal algebra of positive matrices. *Elektron. Informationsverarbeitung und Kybernetik*, 3:39–71, 1967. in Russian.

[Zie98] G. M. Ziegler. *Lectures on Polytopes.* Springer, 1998.

[Zim77] K. Zimmermann. A general separation theorem in extremal algebras. *Ekonom.-Mat. Obzor*, 13(2):179–201, 1977.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 59-70
www.stacs-conf.org

THE REMOTE POINT PROBLEM, SMALL BIAS SPACES, AND EXPANDING GENERATOR SETS

V. ARVIND AND SRIKANTH SRINIVASAN

The Institute of Mathematical Sciences,
C.I.T. Campus, Chennai 600 113, India.

E-mail address, V. Arvind: arvind@imsc.res.in

E-mail address, Srikanth Srinivasan: srikanth@imsc.res.in

Abstract. Using ε-bias spaces over $\mathbb{F}_2$, we show that the Remote Point Problem (RPP), introduced by Alon et al [APY09], has an NC^2 algorithm (achieving the same parameters as [APY09]). We study a generalization of the Remote Point Problem to groups: we replace $\mathbb{F}_2^n$ by $\mathcal{G}^n$ for an arbitrary fixed group $\mathcal{G}$. When $\mathcal{G}$ is Abelian we give an NC^2 algorithm for RPP, again using ε-bias spaces. For nonabelian $\mathcal{G}$, we give a deterministic polynomial-time algorithm for RPP. We also show the connection to construction of expanding generator sets for the group $\mathcal{G}^n$. All our algorithms for the RPP achieve essentially the same parameters as [APY09].

1. Introduction

Valiant, in his celebrated work [V77] on circuit lower bounds for computing linear transformations $A : \mathbb{F}^n \longrightarrow \mathbb{F}^m$ for a field $\mathbb{F}$, initiated the study of rigid matrices. If explicit rigid matrices of certain parameters can be constructed it would result in superlinear lower bounds for logarithmic depth linear circuits over $\mathbb{F}$. This problem and the construction of such rigid matrices has remained elusive for over three decades.

Alon, Panigrahy and Yekhanin [APY09] recently proposed a problem that appears to be of intermediate difficulty. Given a subspace L of $\mathbb{F}_2^n$ by its basis and a number $r \in [n]$ as input, the problem is to compute in deterministic polynomial time a point $v \in \mathbb{F}_2^n$ such that $\Delta(u, v) \geq r$ for all $u \in L$, where $\Delta(u, v)$ is the Hamming distance. They call this the *Remote Point Problem*. The point v is said to be r-far from the subspace L.

1998 ACM Subject Classification: Algorithms and Complexity Theory.

Key words and phrases: Small Bias Spaces, Expander Graphs, Cayley Graphs, Remote Point Problem.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2444

Alon et al [APY09] give a nice polynomial time-bounded (in n) algorithm for computing a $v \in \mathbb{F}_2^n$ that is $c \log n$-far from a given subspace L of dimension $n/2$ and c is a fixed constant. For L such that $\dim(L) = k < n/2$ they give a polynomial-time algorithm for computing a point $v \in \mathbb{F}_2^n$ that is $\frac{cn \log k}{k}$-far from L.

Results of this paper. In [AS09a] we recently investigated the problem of proving circuit lower bounds in the presence of help functions. Specifically, one of the problems we consider is proving lower bounds for constant-depth Boolean circuits which can take a given set of (arbitrary) help functions $\{h_1, h_2, \cdots, h_m\}$ at the input level, where $h_i : \{0,1\}^n \longrightarrow \{0,1\}$ for each i. Proving explicit lower bounds for this model would allow us to separate EXP from the polynomial-time many-one closure of nonuniform AC^0. We show that it suffices to find a polynomial-time solution to the Remote Point Problem for parameters $k = 2^{(\log\log n)^c}$ and $r = \frac{n}{2^{(\log\log n)^d}}$ for all constants c and d. Unfortunately, the parameters of the Alon et al algorithm are inadequate for our application.

However, motivated by this connection, in the present paper we carry out a more detailed study of the Remote Point Problem as an algorithmic question. We briefly summarize our results.

1. The first question we address is whether we can give a deterministic parallel (i.e. NC) algorithm for the problem — Alon et al's algorithm is inherently sequential as it is based on the method of conditional probabilities and pessimistic estimators.

It turns out an element of an ε-bias space for suitably chosen ε is a solution to the Remote Point Problem which gives us an NC algorithm quite easily.

2. Since the RPP for $\mathbb{F}_2^n$ can be solved using small bias spaces, it naturally leads us to address the problem in a more general group-theoretic setting.

In the generalization we study we will replace $\mathbb{F}_2$ with an arbitrary fixed finite group $\mathcal{G}$ such that $|\mathcal{G}| \geq 2$. Hence we will have the n-fold product group $\mathcal{G}^n$ instead of the vector space $\mathbb{F}_2^n$.

Given elements $x = (x_1, x_2, \ldots, x_n), y = (y_1, y_2, \ldots, y_n)$ of $\mathcal{G}^n$, let $\Delta(x,y) = |\{i \mid x_i \neq y_i\}|$. I.e. $\Delta(x,y)$ is the *Hamming distance* between x and y. Furthermore, for $S \subseteq \mathcal{G}^n$, let $\Delta(x,S)$ denote $\min_{y \in S} \Delta(x,y)$.

We now define the *Remote Point Problem (RPP) over a finite group* $\mathcal{G}$. The input is a subgroup $\mathcal{H}$ of $\mathcal{G}^n$, where $\mathcal{H}$ is given by a generating set, and a number $r \in [n]$. The problem is to compute in deterministic polynomial (in n) time an element $x \in \mathcal{G}^n$ such that $\Delta(x,H) > r$. The results we show in this general setting are the following.

(a) The Remote Point Problem over any *Abelian group* $\mathcal{G}$ has an NC^2 algorithm for $r = O(\frac{n \log k}{k})$ and $k \leq n/2$, where $k = \log_{|\mathcal{G}|} |\mathcal{H}|$.

(b) Over an arbitrary group $\mathcal{G}$ the Remote point problem has a polynomial-time algorithm for $r = O(\frac{n \log k}{k})$ and $k \leq n/2$, where $k = \log_{|\mathcal{G}|} |\mathcal{H}|$.

The parallel algorithm stated in part(a) above is based on ε-bias space constructions for finite Abelian groups described in Azar et al [AMN98]. The sequential algorithm stated in part(b) above is a group-theoretic generalization of the Alon et al algorithm for $\mathbb{F}_2^n$ [APY09].

Due to lack of space, some proofs have been omitted. They may be found in the full version which has been published as an ECCC report [AS09b].

2. Preliminaries

Fix a finite group $\mathcal{G}$ such that $|\mathcal{G}| \geq 2$. Given any $x \in \mathcal{G}^n$, let $wt(x)$ denote the number of coordinates i such that $x_i \neq 1$, where 1 is the identity of the group $\mathcal{G}$. By $B(r)$, we will refer to the set of $x \in \mathcal{G}^n$ such that $wt(x) \leq r$. Given a subset S of $\mathcal{G}^n$, $B(S,r)$ will denote the set $S \cdot B(r) = \{sx \mid s \in S, x \in B(r)\}$. Clearly, for any $S \subseteq \mathcal{G}^n$ and any $x \in \mathcal{G}^n$, $x \in B(S,r)$ if and only if $\Delta(x,S) \leq r$. We say that x is *r-close* to S if $x \in B(S,r)$ and *r-far* from S if $x \notin B(S,r)$.

The *Remote Point Problem (RPP) over* $\mathcal{G}$ is defined to be the following algorithmic problem:

> INPUT: A subgroup $\mathcal{H}$ of $\mathcal{G}^n$ (given by its generators) and an $r \in \mathbb{N}$.
> OUTPUT: An $x \in \mathcal{G}^n$ such that $x \notin B(\mathcal{H},r)$.

Clearly, there are inputs to the above problem where no solution can be found. But the input instances of the kind that we will study will clearly have a solution (in fact, a random point of $\mathcal{G}^n$ will be a solution with high probability).

Given a subgroup $\mathcal{H}$ of $\mathcal{G}^n$, denote by $\delta(\mathcal{H})$ the quantity $\log_{|\mathcal{G}|}|\mathcal{H}|$. We will call $\delta(\mathcal{H})$ the *dimension of* $\mathcal{H}$ *in* $\mathcal{G}^n$.

We say that the RPP over $\mathcal{G}$ has a $(k(n), r(n))$-algorithm if there is an efficient algorithm that solves the Remote Point Problem when given as input a subgroup $\mathcal{H}$ of $\mathcal{G}^n$ of dimension at most $k(n)$ and an r that is bounded by $r(n)$. (Here, 'efficient' can correspond to polynomial time or some smaller complexity class.)

A simple counting argument shows that there is a valid solution to the RPP over $\mathcal{G}$ on inputs $(\mathcal{H}, r)$ where $\delta(\mathcal{H}) + r \leq n(1 - \frac{H(r/n)}{\log|G|} - \varepsilon)$, for any fixed $\varepsilon > 0$ (where $H(\cdot)$ denotes the binary entropy function). However, the best known deterministic solution to the RPP – from [APY09] – is a polynomial time $(k, \frac{cn \log k}{k})$-algorithm which works over $\mathbb{F}_2^n$ (i.e, the group $\mathcal{G}$ involved is the additive group of the field $\mathbb{F}_2$).

2.1. Some Group-Theoretic Algorithms

We introduce basic definitions and review some group-theoretic algorithms. Let $\mathrm{Sym}(\Omega)$ denote the group of all permutations on a finite set Ω of size m. In this section we use G, H etc. to denote *permutation groups on* Ω, which are simply subgroups of $\mathrm{Sym}(\Omega)$.

Let G be a subgroup of $\mathrm{Sym}(\Omega)$. For a subset $\Delta \subseteq \Omega$ denote by $G_{\{\Delta\}}$ the *point-wise stabilizer* of Δ. I.e $G_{\{\Delta\}}$ is the subgroup consisting of exactly those elements of G that fix each element of Δ.

Theorem 2.1 (Schreier-Sims)**.** [Lu93]

(1) *If a subgroup G of* $\mathrm{Sym}(\Omega)$ *is given by a generating set as input along with the subset Δ there is a polynomial-time (sequential) algorithm for computing a generator set for $G_{\{\Delta\}}$.*
(2) *If a subgroup G of* $\mathrm{Sym}(\Omega)$ *is given by a generating set as input, then there is a polynomial time algorithm for computing $|G|$.*
(3) *Given as input a permutation $\sigma \in \mathrm{Sym}(\Omega)$ and a generator set for a subgroup G of* $\mathrm{Sym}(\Omega)$*, we can test in deterministic polynomial time if σ is an element of G.*

We are also interested in a special case of this problem which we now define. A subset $\Gamma \subseteq \Omega$ is an *orbit* of G if $\Gamma = \{\sigma(i) \mid \sigma \in G\}$ for some $i \in \Omega$. Any subgroup G of $\mathrm{Sym}(\Omega)$ partitions Ω into orbits (called G-orbits).

For a constant $b > 0$, a subgroup G of $\mathrm{Sym}(\Omega)$ is defined to be a *b-bounded permutation group* if every G-orbit is of size at most b.

In [MC87], McKenzie and Cook studied the parallel complexity of *Abelian* permutation group problems. Specifically, they gave an NC^3 algorithm for testing membership in an Abelian permutation group given by a generator set and for computing the order of an Abelian permutation group. When restricted to b-bounded Abelian permutation groups, the algorithms of [MC87] for these problems are actually NC^2 algorithms. We formally state their result and derive a consequence.

Theorem 2.2 ([MC87])**.** *There is an* NC^2 *algorithm for membership testing in a b-bounded Abelian permutation group G given by a generator set.*

We now consider problems over $\mathcal{G}^n$, for a fixed finite group $\mathcal{G}$. We know from basic group theory that every group $\mathcal{G}$ is a permutation group acting on itself. I.e. every $\mathcal{G}$ can be seen as a subgroup of $\mathrm{Sym}(\mathcal{G})$, where $\mathcal{G}$ acts on itself by left (or right) multiplication. Therefore, $\mathcal{G}^n$ can be easily seen as a permutation group on the set $\Omega = \mathcal{G} \times [n]$ and hence, $\mathcal{G}^n$ can be considered a subgroup of $\mathrm{Sym}(\Omega)$. Furthermore, notice that each subset $\mathcal{G} \times \{i\}$ is an orbit of this group $\mathcal{G}^n$. Hence, $\mathcal{G}^n$ is a b-bounded permutation group contained in $\mathrm{Sym}(\Omega)$, where $b = |\mathcal{G}|$. Finally, if $\mathcal{G}$ is an Abelian group, then so is this subgroup of $\mathrm{Sym}(\Omega)$. We have the following lemma as an easy consequence of Theorem 2.2.

Lemma 2.3. *Let $\mathcal{G}$ be Abelian. There is an* NC^2 *algorithm that takes as input a generator set for some subgroup $\mathcal{H}$ of $\mathcal{G}^n$ and an $x \in \mathcal{G}^n$, and accepts iff $x \in \mathcal{H}$.*

Given any $y = (y_1, y_2, \ldots, y_i) \in \mathcal{G}^i$ with $1 \le i \le n$ and any $S \subseteq \mathcal{G}^n$, let S_y denote the set $\{x \in S \mid x_j = y_j \text{ for } 1 \le j \le i\}$.

Lemma 2.4. *Let $\mathcal{G}$ be any fixed finite group. There is a polynomial time algorithm that takes as input a subgroup $\mathcal{H}$ of $\mathcal{G}^n$, where $\mathcal{H}$ is given by generators, and a $y \in \mathcal{G}^i$ with $1 \le i \le n$, and computes $|\mathcal{H}_y|$.*

Proof. Let $\mathcal{K} = \{(x_1, x_2, \ldots, x_n) \in \mathcal{H} \mid x_1 = x_2 = \cdots = x_i = 1\}$, where 1 denotes the identity element of $\mathcal{G}$. Clearly, $\mathcal{K}$ is a subgroup of $\mathcal{H}$. The set $\mathcal{H}_y$, if nonempty, is simply a coset of $\mathcal{K}$ and thus, we have $|\mathcal{H}_y| = |\mathcal{K}|$. To check if $\mathcal{H}_y$ is nonempty, we consider the map $\pi_i : \mathcal{G}^n \to \mathcal{G}^i$ that projects its input onto its first i coordinates; note that $\mathcal{H}_y$ is nonempty iff the subgroup $\pi_i(\mathcal{H})$ contains y, which can be checked in polynomial time by point (3) of Theorem 2.1 (here, we are identifying $\mathcal{G}^n$ with a subgroup of $\mathrm{Sym}(\mathcal{G} \times [n])$ as above). If $y \notin \pi_i(\mathcal{H})$, the algorithm outputs 0. Otherwise, we have $|\mathcal{H}_y| = |\mathcal{K}|$ and it suffices to compute $|\mathcal{K}|$. But $\mathcal{K}$ is simply the point-wise stabilizer of the set $\mathcal{G} \times [i]$ in $\mathcal{H}$, and hence $|\mathcal{K}|$ can be computed in polynomial time by points (1) and (2) of Theorem 2.1. ■

3. Expanding Cayley Graphs and the Remote Point Problem

Fix a group $\mathcal{G}$ such that $|\mathcal{G}| \geq 2$, and consider an instance of the RPP over $\mathcal{G}$. The main idea that we develop in this section is that if we have a (symmetric) expanding generator set S for the group $\mathcal{G}^n$ with appropriate expansion parameters then for a subgroup $\mathcal{H}$ of $\mathcal{G}^n$ such that $\delta(\mathcal{H}) \leq k$ some element of S will be r-far from H, for suitable k and r.

We review some definitions related to expander graphs (e.g. see the survey of Hoory, Linial, and Wigderson [HLW06]). An undirected multigraph $G = (V, E)$ is an (n, d, α)-graph for $n, d \in \mathbb{N}$ and $\alpha > 0$ if $|V| = n$, the degree of each vertex is d, and the second largest value $\lambda(G)$ from among the absolute values of eigenvalues of $A(G)$ – the adjacency matrix of the graph G – is bounded by αd.

A *random walk* of length $t \in \mathbb{N}$ on an (n, d, α)-graph $G = (V, E)$ is the output of the following random process: a vertex $v_0 \in V$ of picked uniformly at random, and for $0 \leq i < t$, if v_i has been picked, then v_{i+1} is obtained by selecting a neighbour v_{i+1} uniformly at random (i.e a random edge out of v_i is picked, and v_{i+1} is chosen to be the other endpoint of the edge); the output of the process is $(v_0, v_1, \ldots, v_t)$. We now state an important result regarding random walks on expanders (see [HLW06, Theorem 3.6] for details).

Lemma 3.1. *Let $G = (V, E)$ be an (n, d, α)-graph and $B \subseteq V$ with $|B| \leq \beta n$. Then, the probability that a random walk $(v_0, v_1, \ldots, v_t)$ is entirely contained inside B (i.e, $v_i \in B$ for each i) is bounded by $(\beta + \alpha)^t$.*

Let $\mathcal{H}$ be a group and S a *symmetric* multiset of elements from $\mathcal{H}$. I.e. there is a bijection of multisets $\varphi : S \to S$ such that $\varphi(s) = s^{-1}$ for each $s \in S$. We define the Cayley graph $C(\mathcal{H}, S)$ to be the (multi)graph G with vertex set $\mathcal{H}$ and edges of the form (x, xs) for each $x \in \mathcal{H}$ and each $s \in S$; since S is symmetric, we consider $C(\mathcal{H}, S)$ to be an undirected graph by identifying the edges (x, xs) and $(xs, (xs)\varphi(s))$, for each x and s.

We now show a lemma that will help relate generators of expanding Cayley graphs on $\mathcal{G}^n$ and the RPP over $\mathcal{G}$. In what follows, let S be a symmetric multiset of elements from $\mathcal{G}^n$; let G denote the Cayley graph $C(\mathcal{G}^n, S)$; and let N, D denote $|\mathcal{G}|^n$ and $|S|$ (counted with repetitions) respectively.

Lemma 3.2. *Assume S as above is such that G is an (N, D, α)-graph, where $\alpha \leq \frac{1}{n^d}$, for some fixed $d > 0$. Then, given any subgroup $\mathcal{H}$ of $\mathcal{G}^n$ such that $\delta(\mathcal{H}) \leq 2n/3$, we have $\frac{|S \cap \mathcal{H}|}{|S|} \leq \frac{1}{n^{d/2}}$ for large enough n (where the elements of $S \cap \mathcal{H}$ are counted with repetitions).*

Proof. Let $S' = S \cap \mathcal{H}$ and let $\eta = |S'|/|S|$. We want an upper bound on η. Consider a random walk $(x_0, x_1, \ldots, x_t)$ of length t on the graph G (the exact value of t will be fixed later). Let $\mathcal{B}$ denote the following event: there is a $y \in \mathcal{G}^n$ such that all the vertices $x_0, x_1, \ldots, x_t$ are all contained in the coset $y\mathcal{H}$ of $\mathcal{H}$. Let p denote the probability that $\mathcal{B}$ occurs.

We will first lower bound p. At each step of the random walk, a random $s_i \in S$ is chosen and x_{i+1} is set to $x_i s_i$. If these s_i all happen to belong to S', then the cosets $x_i\mathcal{H}$ and $x_{i+1}\mathcal{H}$ are the same for all i and hence, the event $\mathcal{B}$ does occur. Hence, $p \geq \eta^t$.

We now upper bound p. Fix any coset $y\mathcal{H}$ of the subgroup $\mathcal{H}$. Since the dimension of $\mathcal{H}$ in $\mathcal{G}^n$ is bounded by $2n/3$, we have $|y\mathcal{H}| = |\mathcal{H}| \leq |\mathcal{G}|^{2n/3} \leq 2^{-n/3}|\mathcal{G}^n|$. That is, the coset $y\mathcal{H}$ is a very small subset of $\mathcal{G}^n$. Applying Lemma 3.1, we see that the probability that the random walk $(x_0, x_1, \ldots, x_t)$ is completely contained inside this coset is bounded by $(2^{-n/3} + n^{-d})^t \leq \frac{2^t}{n^{dt}}$, for large enough n. As the total number of cosets of $\mathcal{H}$ is bounded by $|\mathcal{G}|^n$, an application of the union bound tells us that p is upper bounded by $|\mathcal{G}|^n \frac{2^t}{n^{dt}} \leq \frac{|\mathcal{G}|^{n+t}}{n^{dt}}$. Setting $t = \frac{2n}{d \log_{|G|} n - 2}$ we see that p is at most $\frac{1}{n^{dt/2}}$.

Putting the upper and lower bounds together, we see that $\eta^t \leq \frac{1}{n^{dt/2}}$ and hence, $\eta \leq \frac{1}{n^{d/2}}$. This completes the proof. ∎

We follow the structure of the algorithm for the RPP over $\mathbb{F}_2$ in [APY09]. We first describe their $(n/2, c\log n)$-algorithm for the RPP, followed by our own algorithm. We then describe how they extend this algorithm to a $(k, \frac{cn \log k}{k})$-algorithm for any $k \leq n/2$; the same procedure works for our algorithm also.

The $(n/2, c\log n)$-algorithm proceeds as follows. On an input instance consisting of a subgroup V (which is a subspace of $\mathbb{F}_2^n$) of dimension at most $n/2$ and an $r \leq c\log n$,

(1) The algorithm first computes a collection of $m = n^{O(c)}$ subspaces $V_1, V_2, \ldots, V_m$, each of dimension at most $2n/3$ such that $B(V, c\log n) \subseteq \bigcup_{i=1}^m V_i$.
(2) The algorithm then finds an $x \in \mathbb{F}_2^n$ such that $x \notin \bigcup_i V_i$. (This is done using a method similar to the method of pessimistic estimators introduced by Raghavan [Rag88].)

Our algorithm will proceed exactly as the above algorithm in the first step. The second step of our algorithm will be different (assuming that the group $\mathcal{G}$ is Abelian). We first state Step 1 of the algorithm of [APY09] in greater generality:

Lemma 3.3. *Let $\mathcal{G}$ be any fixed finite group with $|\mathcal{G}| \geq 2$. For any constant $c > 0$ and large enough n, the following holds. Given any subgroup $\mathcal{H}$ of $\mathcal{G}^n$ such that $\delta(\mathcal{H}) \leq \frac{n}{2}$, there is a collection of $m \leq n^{10c}$ subgroups $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$ such that $B(\mathcal{H}, c\log n) \subseteq \bigcup_{i=1}^m \mathcal{H}_i$, and*

$\delta(\mathcal{H}_i) \leq 2n/3$ for each i. Moreover, there is a logspace algorithm that, when given as input $\mathcal{H}$ as a set of generators, produces generators for the subgroups $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$.

Proof. The proof follows exactly as in [APY09]. We reproduce it here for completeness and to analyze the complexity of the procedure.

Let 1 denote the identity element of $\mathcal{G}$. For each $S \subseteq [n]$, let $\mathcal{G}^n(S)$ denote the subgroup of $\mathcal{G}^n$ consisting of those x such that $x_i = 1$ for each $i \notin S$. Note that $\delta(\mathcal{G}^n(S)) = |S|$. Also note that for each $S \subseteq [n]$, the group $\mathcal{G}^n(S)$ is a normal subgroup; in particular, this implies that the set $\mathcal{K} \cdot \mathcal{G}^n(S)$ is a subgroup of $\mathcal{G}^n$ whenever $\mathcal{K}$ is a subgroup of $\mathcal{G}^n$.

Partition the set $[n]$ into $\ell \leq 10c \log n$ sets of size at most $\lceil \frac{n}{10c \log n} \rceil$ each – we will call these sets $S_1, S_2, \ldots, S_\ell$. For each $A \subseteq [\ell]$ of size $\lceil c \log n \rceil$, let $\mathcal{K}_A$ denote the subgroup $\mathcal{G}^n(\bigcup_{i \in A} S_i)$. Note that the number of such subgroups is at most $2^\ell \leq n^{10c}$. Also, for each A as above, $\delta(\mathcal{K}_A) = |\bigcup_{i \in A} S_i| \leq \left(\frac{n}{10c \log n} + 1\right)(c \log n + 1) < \frac{n}{9}$, for large enough n.

Consider any $x \in B(c \log n)$ (i.e, an element x of $\mathcal{G}^n$ s.t $wt(x) \leq c \log n$). We know that $x \in \mathcal{G}^n(S)$ for some S of size at most $c \log n$. Hence, it can be seen that $x \in \mathcal{G}^n(\bigcup_{i \in A} S_i)$ for some A of size $\lceil c \log n \rceil$; this shows that $B(c \log n) \subseteq \bigcup_A \mathcal{K}_A$. Therefore, we see that $B(\mathcal{H}, c \log n) = \mathcal{H} B(c \log n) \subseteq \bigcup_A \mathcal{H}\mathcal{K}_A$.

For each $A \subseteq [\ell]$ of size $\lceil c \log n \rceil$, let $\mathcal{H}_A$ denote the subgroup $\mathcal{H}\mathcal{K}_A$ (note that this is indeed a subgroup, since $\mathcal{K}_A$ is a normal subgroup). Moreover, the cardinality of this subgroup is bounded by $|\mathcal{H}| \cdot |\mathcal{K}_A| \leq |\mathcal{G}|^{n/2} |\mathcal{G}|^{n/9} < |\mathcal{G}|^{2n/3}$; hence, $\delta(\mathcal{H}_A) \leq 2n/3$. Thus, the collection of subgroups $\{\mathcal{H}_A\}_A$ satisfies all the properties mentioned in the statement of the lemma. That a set of generators for this subgroup can be computed in deterministic logspace – for some suitable choice of $S_1, S_2, \ldots, S_\ell$ – is a routine check from the definition of the subgroups $\{\mathcal{K}_A\}_A$. This completes the proof of the lemma. ■

Using Lemma 3.3, we are able to efficiently "cover" $B(\mathcal{H}, c \log n)$ for any small subgroup $\mathcal{H}$ of $\mathcal{G}^n$ by a union of small subgroups. Therefore, to find a point that is $c \log n$-far from $\mathcal{H}$, it suffices to find a point $x \in \mathcal{G}^n$ not contained in any of the covering subgroups. To do this, we note that if S is a multiset containing elements from $\mathcal{G}^n$ such that $C(\mathcal{G}^n, S)$ is a Cayley graph with good expansion, then S must contain such an element. This is formally stated below.

Lemma 3.4. *For any constant $c > 0$ and large enough $n \in \mathbb{N}$, the following holds. Let S be any multiset of elements of $\mathcal{G}^n$ such that $\lambda(C(\mathcal{G}^n, S)) < \frac{1}{n^{20c}}$. Then, for $m \leq n^{10c}$ and any collection $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$ of subgroups such that $\delta(\mathcal{H}_i) \leq 2n/3$ for each i, there is some $s \in S$ such that $s \notin \bigcup_i \mathcal{H}_i$.*

Proof. The proof follows easily from Lemma 3.2. Given any $i \in [m]$, we know, from Lemma 3.2, that $|S \cap \mathcal{H}_i| < \frac{|S|}{n^{10c}}$ (where the elements of the multisets are counted with repetitions). Hence, $|S \cap \bigcup_i \mathcal{H}_i| \leq \sum_i |S \cap \mathcal{H}_i| < \frac{m|S|}{n^{10c}} \leq |S|$. Therefore, there must be some $s \in S$ such that $s \notin \bigcup_i \mathcal{H}_i$. ■

Therefore, to find a point x that is $c \log n$-far from the subspace $\mathcal{H}$, it suffices to construct an S such that $C(\mathcal{G}^n, S)$ is a sufficiently good expander, find the covering subgroups $\mathcal{H}_i$ ($i \in [m[$), and then to find an $s \in S$ that does not lie in any of the $\mathcal{H}_i$. We follow the above approach to give an efficient parallel algorithm for the RPP in the case that $\mathcal{G}$ is an Abelian group. For arbitrary groups, we show that the method of [APY09] yields a polynomial time algorithm.

4. Remote Point Problem for Abelian Groups

Fix an Abelian group $\mathcal{G}$. Recall that a *character* χ of $\mathcal{G}^n$ is a homomorphism from $\mathcal{G}^n$ to $\mathbb{C}_1^*$, the multiplicative subgroup of the complex numbers of absolute value 1. For $\varepsilon > 0$, a distribution μ over $\mathcal{G}^n$ is said to be ε-biased if, given any non-trivial character χ of $\mathcal{G}^n$, $|\mathbf{E}_{x \sim \mu}[\chi(x)]| \leq \varepsilon$.

A multiset S consisting of elements from $\mathcal{G}^n$ is said to be an *ε-biased space in $\mathcal{G}^n$* if the uniform distribution over S is an ε-biased distribution.

It can be checked that a multiset consisting of $(\frac{n}{\varepsilon})^{O(1)}$ independent, uniformly random elements from $\mathcal{G}^n$ form an ε-biased space with high probability. Explicit ε-biased spaces were constructed for the group $\mathbb{F}_2^n$ by Naor and Naor in [NN93]; further constructions were given by Alon et al. in [AGHP92]. Explicit constructions of ε-biased spaces in $\mathbb{Z}_d^n$ were given by Azar et al. in [AMN98]. We observe that this last construction yields a construction for all Abelian groups $\mathcal{G}^n$, when $\mathcal{G}$ is of constant size. We first state the result of [AMN98] in a form that we will find suitable.

Theorem 4.1. *For any fixed d, there is an* NC^2 *algorithm that does the following. On input n and $\varepsilon > 0$ (both in unary), the algorithm produces a symmetric multiset $S \subseteq \mathbb{Z}_d^n$ of size $O((\frac{n}{\varepsilon})^2)$ such that S is an ε-biased space in $\mathbb{Z}_d^n$.*

Proof. It is easy to see that the ε-biased space construction in [AMN98] can be implemented in deterministic logspace (and hence in NC^2). If the space S obtained is not symmetric, we can consider the multiset that is the disjoint union of S and S^{-1}, which is also easily seen to be ε-biased. ∎

Remark 4.2. We note that the definition of small bias spaces in [AMN98] differs somewhat from our own definition above. But it is easy to see that an ε-bias space in $\mathbb{Z}_d^n$ in the sense of [AMN98] is a $(d\varepsilon)$-bias space according to our definition above.

Remark 4.3. In a recent paper, Meka and Zuckerman [MZ09] observe, as we do below, that the construction of [AMN98] gives small bias spaces for any arbitrary Abelian group $\mathcal{G}$. Nevertheless, we present our own proof of this fact, since the small bias spaces that follow from our proof are of *smaller* size. Specifically, our proof shows how to explicitly construct sample spaces of size $O\left(\frac{n^2}{\varepsilon^2}\right)$, whereas the relevant result in [MZ09] only produces small bias spaces of size $O\left((\frac{n}{\varepsilon})^b\right)$, where b is some constant that depends on $\mathcal{G}$ (and can be as large as $\Omega(\log |\mathcal{G}|)$).

Lemma 4.4. *For any fixed group $\mathcal{G}$, there is an* NC^2 *algorithm which, on input n and $\varepsilon > 0$ in unary, produces a symmetric multiset $S \subseteq \mathcal{G}^n$ of size $O((\frac{n}{\varepsilon})^2)$ such that S is an ε-biased space in $\mathcal{G}^n$.*

Proof. By the Fundamental Theorem of finite Abelian groups, $\mathcal{G} \cong \mathbb{Z}_{d_1} \oplus \mathbb{Z}_{d_2} \oplus \cdots \oplus \mathbb{Z}_{d_k}$, for positive integers $d_1, d_2, \ldots, d_k$ such that $d_1 \mid d_2 \mid \cdots \mid d_k$. Let $\mathcal{G}_0$ denote $\mathbb{Z}_{d_k}^k$. Note that for any $s, t \in \mathbb{N}$, $\mathbb{Z}_s \cong \mathbb{Z}_{st}/\mathbb{Z}_t$. Hence, we see that that $\mathcal{G} \cong \mathcal{G}_0/\mathcal{H}$, where $\mathcal{H}$ is the subgroup $\mathbb{Z}_{e_1} \oplus \mathbb{Z}_{e_2} \oplus \cdots \oplus \mathbb{Z}_{e_k}$, and $e_i = d_k/d_i$ for each $i \in [k]$. Therefore, $\mathcal{G}^n \cong \mathcal{G}_0^n/\mathcal{H}^n$. Let $\pi : \mathcal{G}_0^n \to \mathcal{G}^n$ be the natural onto homomorphism with kernel $\mathcal{H}^n$. Note that π is just the projection map and can easily be computed in NC^2.

Since $\mathcal{G}_0^n \cong \mathbb{Z}_{d_k}^{nk}$, by Theorem 4.1, there is an NC^2 algorithm that constructs a symmetric multiset $S_0 \subseteq \mathcal{G}_0^n$ of size $O(\left(\frac{kn}{\varepsilon}\right)^2)$ such that S_0 is an ε-biased space in $\mathcal{G}_0^n$. We claim that the multiset $S = \pi(S_0)$ is a symmetric ε-biased space in $\mathcal{G}^n$. To see this, consider any non-trivial character χ of $\mathcal{G}^n$; note that $\chi_0 = \chi \circ \pi$ is a non-trivial character of $\mathcal{G}_0^n$. We have

$$\left|\mathop{\mathbf{E}}_{x \sim S}[\chi(x)]\right| = \left|\mathop{\mathbf{E}}_{x_0 \sim S_0}[\chi(\pi(x_0))]\right| = \left|\mathop{\mathbf{E}}_{x_0 \sim S_0}[\chi_0(x)]\right| \leq \varepsilon$$

where the first equality follows from the definition of S, and the last inequality follows from the fact that S_0 is an ε-biased space in $\mathcal{G}_0^n$. Since χ was an arbitrary non-trivial character of $\mathcal{G}^n$, we have proved that S is indeed an ε-biased space in $\mathcal{G}^n$. It is easy to see that S is symmetric. Finally, note that S can be computed in NC^2. This completes the proof. ■

Finally, we mention a well-known connection between small bias spaces in $\mathcal{G}^n$ and Cayley graphs over $\mathcal{G}^n$ (e.g. see Alon and Roichman [AR94]).

Lemma 4.5. *Given any symmetric multiset $S \subseteq \mathcal{G}^n$, the Cayley graph $C(\mathcal{G}^n, S)$ is an $(|\mathcal{G}|^n, |S|, \alpha)$-graph iff S is an α-biased space.*

Lemmas 4.5 and 4.4 have the following easy consequence:

Lemma 4.6. *For any Abelian group $\mathcal{G}$, there is an* NC^2 *algorithm which, on unary inputs n and $\alpha > 0$, produces a symmetric multiset $S \subseteq \mathcal{G}^n$ of size $O((\frac{n}{\alpha})^2)$ such that $C(\mathcal{G}^n, S)$ is a $(|\mathcal{G}|^n, |S|, \alpha)$-graph.*

Putting the above statement together with the results of Section 3, we have the following.

Theorem 4.7. *For any constant $c > 0$, the RPP over $\mathcal{G}$ has an* NC^2 *$(n/2, c\log n)$-algorithm.*

Proof. Let $\mathcal{H}$ denote the input subgroup. By Lemma 3.3, there is a logspace (and hence NC^2) algorithm that computes a collection of $m = n^{O(c)}$ many subgroups $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$ such that $B(\mathcal{H}, c\log n) \subseteq \bigcup_{i=1}^m \mathcal{H}_i$ and $\delta(\mathcal{H}_i) \leq 2n/3$ for each $i \in [m]$. Now, fix any multiset $S \subseteq \mathcal{G}^n$ such that the Cayley graph $C(\mathcal{G}^n, S)$ is a $(|\mathcal{G}|^n, |S|, \alpha)$-graph, where $\alpha = \frac{1}{2n^{20c}}$; by Lemma 4.6, such an S can be constructed in NC^2. It follows from Lemma 3.4 that there is some $s \in S$ such that $s \notin \bigcup_{i=1}^m \mathcal{H}_i$. Finally, by Lemma 2.3, there is an NC^2 algorithm to test if each $s \in S$ belongs to $\mathcal{H}_i$, for any $i \in [m]$. Hence, we can find out (in parallel) exactly which $s \in S$ do not belong to any of the $\mathcal{H}_i$ and output one of them. The output element s is surely $c\log n$-far from $\mathcal{H}$. ■

Let $\mathcal{G}$ be Abelian. We observe that a method of [APY09], coupled with Theorem 4.7, yields an efficient $(k, \frac{cn\log k}{k})$-algorithm for any constant $c > 0$, and $k \leq n/2$.

Theorem 4.8. *Let $c > 0$ be any constant. If $\mathcal{G}$ is an Abelian group, then the RPP over $\mathcal{G}$ has an* NC^2 *$(k, \frac{cn\log k}{k})$-algorithm for any $k \leq n/2$.*

Proof. Given as input a subgroup $\mathcal{H}$ such that $\delta(\mathcal{H}) = k \leq n/2$, the algorithm partitions $[n]$ as $[n] = \bigcup_{i=1}^{m} T_i$, where $2k \leq |T_i| < 4k$ for each i; note that $m \geq n/4k$. Let $\mathcal{H}_i$ denote the subgroup obtained when $\mathcal{H}$ is projected onto the coordinates in T_i. Since $\delta(\mathcal{H}_i) \leq k \leq |T_i|/2$, we can, by Theorem 4.7, efficiently find a point $x_i \in \mathcal{G}^{|T_i|}$ that is at least $4c\log k$-far from $\mathcal{H}_i$. Putting these x_i together in the natural way, we obtain an $x \in \mathcal{G}^n$ that is $\frac{cn\log k}{k}$-far from the subgroup $\mathcal{H}$.

Since $\mathcal{G}$ is Abelian, using the algorithm of Theorem 4.7, the x_i can all be computed in parallel in NC^2. Hence, the entire procedure can be performed in NC^2. ■

5. RPP over General Groups

Let $\mathcal{G}$ denote some fixed finite group. We can generalize the polynomial-time algorithm of [APY09], described for $\mathbb{F}_2$, to compute a point $x \in \mathcal{G}^n$ that is $c\log n$-far from a given input subgroup $\mathcal{H}$ such that $\delta(\mathcal{H}) \leq n/2$. We only state this result below and refer the interested reader to the full version [AS09b] for details.

Theorem 5.1. *For any constant $c > 0$, the RPP over $\mathcal{G}$ has a polynomial time $(n/2, c\log n)$-algorithm.*

Analogous to Theorem 4.8, we have the following solution to RPP for general groups.

Theorem 5.2. *Let $c > 0$ be any constant. For any $\mathcal{G}$, the RPP over $\mathcal{G}$ has a polynomial time $(k, \frac{cn\log k}{k})$-algorithm for any $k \leq n/2$.*

Proof. The construction is exactly the same as in the proof of Theorem 4.8. The only difference is that we will apply the algorithm of Theorem 5.1. In this case, the x_i can all be found in deterministic polynomial time. Hence, the entire procedure gives us a polynomial-time algorithm. ■

6. Limitations of expanding sets

In the previous sections, we have shown how generators for expanding Cayley graphs on $\mathcal{G}^n$, where $\mathcal{G}$ is a fixed finite group, can help solve the RPP over $\mathcal{G}$. In particular, we have the following easy consequence of Lemmas 3.3 and 3.4.

Corollary 6.1. *For any constant $c > 0$, large enough n, and any symmetric multiset $S \subseteq \mathcal{G}^n$ such that $\lambda(C(\mathcal{G}^n, S)) < \frac{1}{n^{20c}}$, the following holds. If $\mathcal{H}$ is any subgroup of $\mathcal{G}^n$ such that $\delta(\mathcal{H}) \leq n/2$, there is some $s \in S$ such that $s \notin B(\mathcal{H}, c\log n)$.*

It makes sense to ask if the parameters in Corollary 6.1 are far from optimal. Is it true that any polynomial-sized symmetric multiset $S \subseteq \mathcal{G}^n$ with good enough expansion properties is $\omega(\log n)$-far from every subgroup of dimension at most $n/2$? We can show that this is not true. Formally, we can prove:

Theorem 6.2. *For any constant $c > 0$ and large enough n, there is a symmetric multiset $S \subseteq \mathbb{F}_2^n$ such that $\lambda(C(\mathbb{F}_2^n, S)) \leq \frac{1}{n^c}$ but there is a subspace L of dimension $n/2$ such that $S \subseteq B(L, 20c \log n)$.*

It is well known that for any family of d-regular multigraphs G $\lambda(G) = \Omega(1/\sqrt{d})$ (see e.g. [HLW06, Theorem 5.3]). As a consequence of this lower bound it follows for any fixed group $\mathcal{G}$ and any multiset $S \subseteq \mathcal{G}^n$ that $\lambda(C(\mathcal{G}, S)) = \Omega(1/\sqrt{|S|})$. Hence, the above theorem tells us that just the expansion properties of $C(\mathbb{F}_2^n, S)$ for any poly(n)-sized S are not sufficient to guarantee $\omega(\log n)$-distance from every subspace of dimension $n/2$. The proof of the above statement can be found in the full version [AS09b].

7. Discussion

For the remote point problem over an Abelian group $\mathcal{G}$, we have shown how expanding generating sets for Cayley graphs of $\mathcal{G}^n$ can be used to obtain deterministic NC^2 algorithms. A natural question is whether we can obtain a similar algorithm for non-Abelian $\mathcal{G}$. Note that Lemma 3.4 holds in the non-Abelian setting too. Hence, in order to obtain an NC^2-algorithm for the RPP over arbitrary non-Abelian $\mathcal{G}$ along the lines of our algorithm for Abelian groups, we need to be able to check (in NC^2) for membership in $\mathcal{G}^n$, and we need to be able to construct small multisets S of $\mathcal{G}^n$ such that $C(\mathcal{G}^n, S)$ has sufficiently good expansion properties. Luks' work [Lu86] yields an NC^4 test for membership in $\mathcal{G}^n$ for arbitrary $\mathcal{G}$. Building on that, there is also an NC^2 membership test for $\mathcal{G}^n$ [AKV05]. However, we are unable to compute a (good enough) expanding generator set for the group $\mathcal{G}^n$ in deterministic NC or even in deterministic polynomial time.

Acknowledgements

We are grateful to Noga Alon and Sergey Yekhanin for interesting comments. In particular, Alon pointed out to us that Lemma 3.2 has an alternative proof using the expander mixing lemma. We thank the anonymous referees for their comments and suggestions.

References

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992.

[AKV05] V. Arvind, Piyush P. Kurur, T. C. Vijayaraghavan. Bounded Color Multiplicity Graph Isomorphism is in the #L Hierarchy. In *IEEE Conference on Computational Complexity 2005*: 13-27.

[APY09] Noga Alon, Rina Panigrahy, and Sergey Yekhanin. Deterministic approximation algorithms for the nearest codeword problem. In *APPROX-RANDOM*, pages 339–351, 2009.

[AR94] Noga Alon, Yuval Roichman. Random Cayley Graphs and Expanders. *Random Structures and Algorithms,* 5(2): 271-285 (1994).

[AS09a] V. Arvind and Srikanth Srinivasan. Circuit Complexity, Help Functions and the Remote point problem. manuscript.

[AS09b] V. Arvind and Srikanth Srinivasan. The Remote Point Problem, Small Bias Spaces, and Expanding Generator Sets ECCC Report TR09-105. Can be found at `http://eccc.hpi-web.de/report/2009/105/`

[AMN98] Yossi Azar, Rajeev Motwani, and Joseph Naor. Approximating probability distributions using small sample spaces. *Combinatorica*, 18(2):151–171, 1998.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc. (N.S)*, 43:439–561, 2006.

[Lu86] Eugene M. Luks. Parallel algorithms for permutation groups and graph isomorphism. In *FOCS*, pages 292–302, 1986.

[Lu93] Eugene M. Luks. Permutation groups and polynomial time computation. *Groups and Computation I,* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 11, 139-174, 1993.

[MC87] Pierre McKenzie and Stephen Cook. The parallel complexity of Abelian permutation group problems. *SIAM Journal on Computing*, 16(5):880-909, 1987.

[MZ09] Raghu Meka and David Zuckerman. Small-Bias Spaces for Group Products. *APPROX-RANDOM 2009*: 658-672.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[Rag88] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130 – 143, 1988.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

[V77] Leslie G. Valiant. Graph-Theoretic Arguments in Low-Level Complexity. *Proceedings Mathematical Foundations of Computer Science,* LNCS vol. 53: 162-176, Springer 1977.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 71-82
www.stacs-conf.org

EVASIVENESS AND THE DISTRIBUTION OF PRIME NUMBERS

LÁSZLÓ BABAI [1,2] AND ANANDAM BANERJEE [3] AND RAGHAV KULKARNI [1] AND VIPUL NAIK [1]

[1] University of Chicago, Chicago, IL, USA.

[3] Northeastern University, Boston, MA, USA.

Abstract. A Boolean function on N variables is called *evasive* if its decision-tree complexity is N. A sequence B_n of Boolean functions is *eventually evasive* if B_n is evasive for all sufficiently large n.

We confirm the eventual evasiveness of several classes of monotone graph properties under widely accepted number theoretic hypotheses. In particular we show that Chowla's conjecture on Dirichlet primes implies that (a) for any graph H, "forbidden subgraph H" is eventually evasive and (b) all nontrivial monotone properties of graphs with $\leq n^{3/2-\epsilon}$ edges are eventually evasive. (n is the number of vertices.)

While Chowla's conjecture is not known to follow from the Extended Riemann Hypothesis (ERH, the Riemann Hypothesis for Dirichlet's L functions), we show (b) with the bound $O(n^{5/4-\epsilon})$ under ERH.

We also prove unconditional results: (a$'$) for any graph H, the query complexity of "forbidden subgraph H" is $\binom{n}{2}-O(1)$; (b$'$) for some constant $c > 0$, all nontrivial monotone properties of graphs with $\leq cn \log n + O(1)$ edges are eventually evasive.

Even these weaker, unconditional results rely on deep results from number theory such as Vinogradov's theorem on the Goldbach conjecture.

Our technical contribution consists in connecting the topological framework of Kahn, Saks, and Sturtevant (1984), as further developed by Chakrabarti, Khot, and Shi (2002), with a deeper analysis of the orbital structure of permutation groups and their connection to the distribution of prime numbers. Our unconditional results include stronger versions and generalizations of some result of Chakrabarti et al.

1. Introduction

1.1. The framework

A *graph property* P_n of n-vertex graphs is a collection of graphs on the vertex set $[n] = \{1, \ldots, n\}$ that is invariant under relabeling of the vertices. A property P_n is called *monotone* (decreasing) if it is preserved under the deletion of edges. The trivial graph properties are the empty set and the set of all graphs. A class of examples are the *forbidden*

1998 ACM Subject Classification: F.2.2, F.1.1, F.1.3.

Key words and phrases: Decision tree complexity, evasiveness, graph property, group action, Dirichlet primes, Extended Riemann Hypothesis.

[2] Partially supported by NSF Grant CCF-0830370.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2445

subgraph properties: for a fixed graph H, let Q_n^H denote the class of n-vertex graphs that do not contain a (not necessarily induced) subgraph isomorphic to H.

We view a set of labeled graphs on n vertices as a Boolean function on the $N = \binom{n}{2}$ variables describing adjacency. A Boolean function on N variables is *evasive* if its deterministic query (decision-tree) complexity is N.

The long-standing Aanderaa-Rosenberg-Karp conjecture asserts that *every nontrivial monotone graph property is evasive.* The problem remains open even for important special classes of monotone properties, such as the forbidden subgraph properties.

1.2. History

In this note, n always denotes the number of vertices of the graphs under consideration.

Aanderaa and Rosenberg (1973) [17] conjectured a lower bound of $\Omega(n^2)$ on the query complexity of monotone graph properties. Rivest and Vuillemin (1976) [19] verified this conjecture, proving an $n^2/16$ lower bound. Kleitman and Kwiatkowski (1980) [10] improved this to $n^2/9$. Karp conjectured that nontrivial monotone graph properties were in fact evasive. We refer to this statement as the Aanderaa-Rosenberg-Karp (ARK) conjecture.

In their seminal paper, Kahn, Saks, and Sturtevant [11] observe that non-evasiveness of monotone Boolean functions has strong topological consequences (contracibility of the associated simplicial complex). They then use results of R. Oliver about fixed points of group actions on such complexes to verify the ARK conjecture when n is a prime-power. As a by-product, they improve the lower bound for general n to $n^2/4$.

Since then, the topological approach of [11] has been influential in solving various interesting special cases of the ARK conjecture. Yao (1988) [25] proves that non-trivial monotone properties of bipartite graphs with a given partition (U, V) are evasive (require $|U||V|$ queries). Triesch (1996) [22] shows (in the original model) that any monotone property of bipartite graphs (all the graphs satisfying the property are bipartite) is evasive. Chakrabarti, Khot, and Shi (2002) [3] introduce important new techniques which we use; we improve over several of their results (see Section 1.4).

1.3. Prime numbers in arithmetic progressions

Dirichlet's Theorem (1837) (cf. [5]) asserts that if $\gcd(a, m) = 1$ then there exist infinitely many primes $p \equiv a \pmod{m}$. Let $p(m, a)$ denote the smallest such prime p. Let $p(m) = \max\{p(m, a) \mid \gcd(a, m) = 1\}$. Linnik's celebrated theorem (1947) asserts that $p(m) = O(m^L)$ for some absolute constant L (cf. [16, Chap. V.]). Heath-Brown [9] shows that $L \leq 5.5$. Chowla [4] observes that under the Extended Riemann Hypothesis (ERH) we have $L \leq 2 + \epsilon$ for all $\epsilon > 0$ and conjectures that $L \leq 1 + \epsilon$ suffices:

Conjecture 1.1 (S. Chowla [4])**.** For every $\epsilon > 0$ and every m we have $p(m) = O(m^{1+\epsilon})$.

This conjecture is widely believed; in fact, number theorists suggest as plausible the stronger form $p(m) = O(m(\log m)^2)$ [8]. Turán [23] proves the tantalizing result that for almost all a we have $p(m, a) = O(m \log m)$.

Let us call a prime p an *ϵ-near Fermat prime* if there exists an $s \geq 0$ such that $2^s \mid p - 1$ and $\frac{p-1}{2^s} \leq p^\epsilon$.

We need the following weak form of Chowla's conjecture:

Conjecture 1.2 (Weak Chowla Conjecture)**.** For every $\epsilon > 0$ there exist infinitely many ϵ-near Fermat primes.

In other words, the weak conjecture says that for every ϵ, for infinitely many values of s we have $p(2^s, 1) < (2^s)^{1+\epsilon}$.

1.4. Main results

For a graph property P we use P_n to denote the set of graphs on vertex set $[n]$ with property P. We say that P is *eventually evasive* if P_n is evasive for all sufficiently large n.

Our first set of results states that the "forbidden subgraph" property is "almost evasive" under three different interpretations of this phrase.

Theorem 1.3 (Forbidden subgraphs)**.** *For all graphs H, the forbidden subgraph property Q_n^H (a) is eventually evasive, assuming the Weak Chowla Conjecture; (b) is evasive for almost all n (unconditionally); and (c) has query complexity $\binom{n}{2} - O(1)$ for all n (unconditionally).*

Part (b) says the asymptotic density of values of n for which the problem is not evasive is zero. Part (c) improves the bound $\binom{n}{2} - O(n)$ given in [3]. Parts (a) and (c) will be proved in Section 3. We defer the proof of part (b) to the journal version.

The term "monotone property of graphs with $\leq m$ edges" describes a monotone property that fails for all graphs with more than m edges.

Theorem 1.4 (Sparse graphs)**.** *All nontrivial monotone properties of graphs with at most $f(n)$ edges are eventually evasive, where (a) under Chowla's Conjecture, $f(n) = n^{3/2-\epsilon}$ for any $\epsilon > 0$; (b) under ERH, $f(n) = n^{5/4-\epsilon}$; and (c) unconditionally, $f(n) = cn \log n$ for some constant $c > 0$. (d) Unconditionally, all nontrivial monotone properties of graphs with no cycle of length greater than $(n/4)(1-\epsilon)$ are eventually evasive (for all $\epsilon > 0$).*

Part (c) of Theorem 1.4 will be proved in Section 4. Parts (a) and (b) follow in Section 5. The proof of part (d) follows along the lines of part (c); we defer the details to the journal version of this paper.

We note that the proofs of the unconditional results (c) and (d) in Theorem 1.4 rely on Haselgrove's version [7] of Vinogradov's Theorem on Goldbach's Conjecture (cf. Sec. 4.2).

Recall that a *topological subgraph* of a graph G is obtained by taking a subgraph and replacing any induced path $u - \cdots - v$ in the subgraph by an edge $\{u, v\}$ (repeatedly) and deleting parallel edges. A *minor* of a graph is obtained by taking a subgraph and contracting edges (repeatedly). If a class of graphs is closed under taking minors then it is also closed under taking topological subgraphs but not conversely; for instance, graphs with maximum degree ≤ 3 are closed under taking toopological subgraphs but every graph is a minor of a regular graph of degree 3.

Corollary 1.5 (Excluded topological subgraphs)**.** *Let P be a nontrivial class of graphs closed under taking topological subgraphs. Then P is eventually evasive.*

This unconditional result extends one of the results of Chakrabarti et al. [3], namely, that nontrival classes of graphs closed under taking minors is eventually evasive.

Corollary 1.5 follows from part (c) of Theorem 1.4 in the light of Mader's Theorem which states that if the average degree of a graph G is greater than $2^{\binom{k+1}{2}}$ then it contains a topological K_k [13, 14].

Theorem 1.4 suggests a new stratification of the ARK Conjecture. For a monotone (decreasing) graph property P_n, let

$$\dim(P_n) := \max\{|E(G)| - 1 \mid G \in P_n\}.$$

We can now restate the ARK Conjecture:

Conjecture 1.6. If P_n is a non-evasive, non-empty, monotone decreasing graph property then $\dim(P_n) = \binom{n}{2} - 1$.

2. Preliminaries

2.1. Group action

For the basics of group theory we refer to [18]. All groups in this paper are finite. For groups Γ_1, Γ_2 we use $\Gamma_1 \leq \Gamma_2$ to denote that Γ_1 is a subgroup; and $\Gamma_1 \triangleleft \Gamma_2$ to denote that Γ_1 is a (not necessarily proper) normal subgroup. We say that Γ is a p-group if $|\Gamma|$ is a power of the prime p.

For a set Ω called the "permutation domain," let $\mathrm{Sym}(\Omega)$ denote the *symmetric group* on Ω, consisting of the $|\Omega|!$ permutations of Ω. For $\Omega = [n] = \{1, \dots, n\}$, we set $\Sigma_n = \mathrm{Sym}([n])$. For a group Γ, a homomorphism $\varphi : \Gamma \to \mathrm{Sym}(\Omega)$ is called a Γ-*action* on Ω. The action is *faithful* if $\ker(\varphi) = \{1\}$. For $x \in \Omega$ and $\gamma \in \Gamma$ we denote by x^γ the image of x under $\varphi(\gamma)$. For $x \in \Omega$ we write $x^\Gamma = \{x^\gamma : \gamma \in \Gamma\}$ and call it the *orbit* of x under the Γ-action. The orbits partition Ω.

Let $\binom{\Omega}{t}$ denote the set of t-subsets of Ω. There is a natural induced action $\mathrm{Sym}(\Omega) \to \mathrm{Sym}(\binom{\Omega}{t})$ which also defines a natural Γ-action on $\binom{\Omega}{t}$. We denote this action by $\Gamma^{(t)}$. Similarly, there is a natural induced Γ-action on $\Omega \times \Omega$. The orbits of this action are called the *orbitals* of Γ. We shall need the undirected version of this concept; we shall call the orbits of the Γ-action on $\binom{\Omega}{2}$ the *u-orbitals* (undirected orbitals) of the Γ-action.

By an action of the group Γ on a structure $\mathfrak{X}$ such as a group or a graph or a simplicial complex we mean a homomorphism $\Gamma \to \mathrm{Aut}(\mathfrak{X})$ where $\mathrm{Aut}(\mathfrak{X})$ denotes the automorphism group of $\mathfrak{X}$.

Let Γ and Δ be groups and let $\psi : \Delta \to \mathrm{Aut}(\Gamma)$ be a Δ-action on Γ. These data uniquely define a group $\Theta = \Gamma \rtimes \Delta$, the *semidirect product* of Γ and Δ with respect to ψ. This group has order $|\Theta| = |\Gamma||\Delta|$ and has the following properites: Θ has two subgroups $\Gamma^* \cong \Gamma$ and $\Delta^* \cong \Delta$ such that $\Gamma^* \triangleleft \Theta$; $\Gamma^* \cap \Delta^* = \{1\}$; and $\Theta = \Gamma^*\Delta^* = \{\gamma\delta \mid \gamma \in \Gamma^*, \delta \in \Delta^*\}$. Moreover, identifying Γ with Γ^* and Δ with Δ^*, for all $\gamma \in \Gamma$ and $\delta \in \Delta$ we have $\gamma^{\psi(\delta)} = \delta^{-1}\gamma\delta$.

Θ can be defined as the set $\Delta \times \Gamma$ under the group operation

$$(\delta_1, \gamma_1)(\delta_2, \gamma_2) = (\delta_1\delta_2, \gamma_1^{\psi(\delta_2)}\gamma_2) \qquad (\delta_i \in \Delta, \gamma_i \in \Gamma).$$

For more on semidirect products, which we use extensively, see [18, Chap. 7].

The group $\mathrm{AGL}(1, q)$ of affine transformations $x \mapsto ax+b$ of $\mathbb{F}_q$ ($a \in \mathbb{F}_q^\times$, $b \in \mathbb{F}_q$) acts on $\mathbb{F}_q$. For each $d \mid q-1$, $\mathrm{AGL}(1, q)$ has a unique subgroup of order qd; we call this subgroup $\Gamma(q, d)$. We note that $\mathbb{F}_q^+ \triangleleft \Gamma(q, d)$ and $\Gamma(q, d)/\mathbb{F}_q^+$ is cyclic of order d and is isomorphic to a subgroup Δ of $\mathrm{AGL}(1, q)$; $\Gamma(q, d)$ can be described as a *semidirect product* $(\mathbb{F}_q^+) \rtimes \Delta$.

2.2. Simplicial complexes and monotone graph properties

An *abstract simplicial complex* $\mathcal{K}$ on the set Ω is a subset of the power-set of Ω, closed under subsets: if $B \subset A \in \mathcal{K}$ then $B \in \mathcal{K}$. The elements of $\mathcal{K}$ are called its *faces*. The *dimension* of $A \in \mathcal{K}$ is $\dim(A) = |A| - 1$; the dimension of $\mathcal{K}$ is $\dim(\mathcal{K}) = \max\{\dim(A) \mid A \in \mathcal{K}\}$. The *Euler characteristic* of $\mathcal{K}$ is defined as

$$\chi(\mathcal{K}) := \sum_{A\in\mathcal{K}, A\neq\emptyset} (-1)^{\dim(A)}.$$

Let $[n] := \{1, 2, \ldots, n\}$ and $\Omega = \binom{[n]}{2}$. Let P_n be a subset of the power-set of Ω, i. e., a set of graphs on the vertex set $[n]$. We call P_n a *graph property* if it is invariant under the induced action $\Sigma_n^{(2)}$. We call this graph property *monotone decreasing* if it is closed under subgraphs, i. e., it is a simplicial complex. We shall omit the adjective "decreasing."

2.3. Oliver's Fixed Point Theorem

Let $\mathcal{K} \subseteq 2^{\Omega}$ be an abstract simplicial complex with a Γ-action. The *fixed point complex* $\mathcal{K}_\Gamma$ action is defined as follows. Let $\Omega_1, \ldots, \Omega_k$ be the Γ-orbits on Ω. Set

$$\mathcal{K}_\Gamma := \{S \subseteq [k] \mid \bigcup_{i\in S} \Omega_i \in \mathcal{K}\}.$$

We say that a group Γ satisfies **Oliver's condition** if there exist (not necessarily distinct) primes p, q such that Γ has a (not necessarily proper) chain of subgroups $\Gamma_2 \lhd \Gamma_1 \lhd \Gamma$ such that Γ_2 is a p-group, Γ_1/Γ_2 is cyclic, and Γ/Γ_1 is a q-group.

Theorem 2.1 (Oliver [15]). *Assume the group Γ satisfies Oliver's condition. If Γ acts on a nonempty contractible simplicial complex $\mathcal{K}$ then*

$$\chi(\mathcal{K}_\Gamma) \equiv 1 \pmod{q}. \tag{2.1}$$

In particular, such an action must always have a nonempty invariant face.

2.4. The KSS approach and the general strategy

The topological approach to evasiveness, initiated by Kahn, Saks, and Sturtevant, is based on the following key observation.

Lemma 2.2 (Kahn-Saks-Sturtevant [11]). *If P_n is a non-evasive graph property then P_n is contractible.*

Kahn, Saks, and Sturtevant recognized that Lemma 2.2 brought Oliver's Theorem to bear on evasiveness. The combination of Lemma 2.2 and Theorem 2.1 suggests the following general strategy, used by all authors in the area who have employed the topological method, including this paper: We find primes p, q, a group Γ satisfying Oliver's condition with these primes, and a Γ-action on P_n, such that $\chi(P_n) \equiv 0 \pmod{q}$. By Oliver's Theorem and the KSS Lemma this implies that P_n is evasive. The novelty is in finding the right Γ.

KSS [11] made the assumption that n is a prime power and used as $\Gamma = \mathrm{AGL}(1, n)$, the group of affine transformations $x \mapsto ax + b$ over the field of order n. While we use subgroups of such groups as our building blocks, the attempt to combine these leads to hard problems on the distribution of prime numbers.

Regarding the "forbidden subgraph" property, Chakrabarti, Khot, and Shi [3] built considerable machinery which we use. Our conclusions are considerably stronger than theirs; the additional techniques involved include a study of the orbitals of certain metacyclic groups, a universality property of cyclotomic graphs derivable using Weil's character sum estimates, plus the number theoretic reductions indicated.

For the "sparse graphs" result (Theorem 1.4) we need Γ such that all u-orbitals of Γ are large and therefore $(P_n)_\Gamma = \{\emptyset\}$.

In both cases, we are forced to use rather large building blocks of size q, say, where q is a prime such that $q-1$ has a large divisor which is a prime for Theorem 1.4 and a power of 2 for Theorem 1.3.

3. Forbidden subgraphs

In this section we prove parts (a) and (c) of Theorem 1.3.

3.1. The CKS condition

A *homomorphism* of a graph H to a graph H' is a map $f : V(H) \to V(H')$ such that $(\forall x, y \in V(H))(\{x,y\} \in E(H) \Rightarrow \{f(x), f(y)\} \in E(H'))$. (In particular, $f^{-1}(x')$ is an independent set in H for all $x' \in V(H')$.) Let $Q_r^{[[H]]}$ be the set of those H' with $V(H') = [r]$ that do not admit an $H \to H'$ homomorphism. Let further $T_H := \min\{2^{2^t} - 1 \mid 2^{2^t} \geq h\}$ where h denotes the number of vertices of H. The following is the main lemma of Chakrabarti, Khot, and Shi [3].

Lemma 3.1 (Chakrabarti et al. [3]). *If* $r \equiv 1 \pmod{T_H}$ *then* $\chi(Q_r^{[[H]]}) \equiv 0 \pmod 2$.

3.2. Cliques in generalized Paley graphs

Let q be an odd prime power and d an even divisor of $q-1$. Consider the graph $P(q,d)$ whose vertex set is $\mathbb{F}_q$ and the adjacency between the vertices is defined as follows: $i \sim j \iff (i-j)^d = 1$. $P(q,d)$ is called a *generalized Paley graph.*

Lemma 3.2. *If* $(q-1)/d \leq q^{1/(2h)}$ *then* $P(q,d)$ *contains a clique on* h *vertices.*

This follows from the following lemma which in turn can be proved by a routine application of Weil's character sum estimates (cf. [1]).

Lemma 3.3. *Let* $a_1, \dots, a_t$ *be distinct elements of the finite field* $\mathbb{F}_q$. *Assume* $\ell \mid q-1$. *Then the number of solutions* $x \in \mathbb{F}_q$ *to the system of equations* $(a_i + x)^{(q-1)/\ell} = 1$ *is* $\frac{q}{\ell^t} \pm t\sqrt{q}$. ■

Let $\Gamma(q,d)$ be the subgroup of order qd of $\mathrm{AGL}(1,q)$ defined in Section 2.1.

Observation 3.4. Each u-orbital of $\Gamma(q,d)$ is isomorphic to $P(q,d)$. ■

Corollary 3.5. *If* $\frac{q-1}{d} \leq q^{1/(2h)}$ *then each* u*-orbital of* $\Gamma(q,d)$ *contains a clique of size* h*.*

3.3. ϵ-near-Fermat primes

The numbers in the title were defined in Section 1.3. In this section we prove Theorem 1.3, part (a).

Theorem 3.6. *Let* H *be a graph on* h *vertices. If there are infinitely many* $\frac{1}{2h}$*-near-Fermat primes then* Q_n^H *is eventually evasive.*

Proof. Fix an odd prime $p \equiv 2 \pmod{T_H}$ such that $p \geq |H|$. If there are infinitely many $\frac{1}{2h}$-near-Fermat primes then infinitely many of them belong to the same residue class mod p, say $a + \mathbb{Z}p$. Let q_i be the i-th $\frac{1}{2h}$-near-Fermat prime such that $q_i \geq p$ and $q_i \equiv a \pmod p$. Let $r' = na^{-1} \pmod p$ and $k' = \sum_{i=1}^{r'} q_i$. Then $k' \equiv n \pmod p$ and therefore $n = pk + k'$ for some k.

Now in order to use Lemma 3.1, we need to write n as a sum of r terms where $r \equiv 1 \pmod{T_H}$. We already have r' of these terms; we shall choose each of the remaining $r - r'$ terms to be p or p^2. If there are t terms equal to p^2 then this gives us a total of $r = t + (k - tp) + r'$ terms, so we need $t(p-1) \equiv k + r' \pmod{T_H}$. By assumption, $p - 1 \equiv 1 \pmod{T_H}$; therefore such a t exists; for large enough n, it will also satisfy the constraints $0 \leq t \leq k/p$,

Let now

$$\Lambda_1 := \left((\mathbb{F}_{p^2}^{+})^t \times (\mathbb{F}_p^{+})^{k-tp}\right) \rtimes \mathbb{F}_{p^2}^{\times}$$

acting on $[pk]$ with t orbits of size p^2 and $k - pt$ orbits of size p as follows: on an orbit of size p^i $(i = 1, 2)$ the action is $\mathrm{AGL}(1, p^i)$. The additive groups act independently, with a single multiplicative action on top. $\mathbb{F}_{p^2}^{\times}$ acts on $\mathbb{F}_p^{+}$ through the group homomorphism $\mathbb{F}_{p^2}^{\times} \to \mathbb{F}_p^{\times}$ defined by the map $x \mapsto x^{p-1}$. Let B_j denote an orbit of Λ_1 on $[kp]$. Now the orbit of any pair $\{u, v\} \in \binom{B_j}{2}$ is a clique of size $|B_j| \geq p \geq h$, therefore a Λ_1-invariant graph cannot contain an intra-cluster edge.

Let d_i be the largest power of 2 that divides $q_i - 1$. Let C_i be the subgroup of $\mathbb{F}_{q_i}^{\times}$ of order d_i. Let $\Lambda_2 := \prod_{i=1}^{r'} \Gamma(q_i, d_i)$, acting on $[k']$ with r' orbits of sizes $q_1, \ldots, q_{r'}$ in the obvious manner.

From Lemma 3.2 we know that the orbit of any $\{u, v\} \in \binom{[q_i]}{2}$ must contain a clique of size h. Hence, an invariant graph cannot contain any intra-cluster edge.

Overall, let $\Gamma := \Lambda_1 \times \Lambda_2$, acting on $[n]$. Since $q_i \geq p$, we have $\gcd(q_i, p^2 - 1) = 1$. Thus, Γ is a "2-group extension of a cyclic extension of a p-group" and therefore satisfies Oliver's Condition (stated before Theorem 2.1). Hence, assuming Q_n^H is non-evasive, Lemma 2.2 and Theorem 2.1 imply

$$\chi((Q_n^H)_\Gamma) \equiv 1 \pmod 2.$$

On the other hand, we claim that the fixed-point complex $(Q_n^H)_\Gamma$ is isomorphic to $Q_r^{[[H]]}$. The (simple) proof goes along the lines of Lemma 4.2 of [3]. Thus, by Lemma 3.1 we have $\chi(Q_r^{[[H]]}) \equiv 0 \pmod 2$, a contradiction. ■

3.4. Unconditionally, Q_n^H is only $O(1)$ away from being evasive

In this section, we prove part (c) of Theorem 1.3.

Theorem 3.7. *For every graph H there exists a number C_H such that the query complexity of Q_n^H is $\geq \binom{n}{2} - C_H$.*

Proof. Let h be the number of vertices of H. Let p be the smallest prime such that $p \geq h$ and $p \equiv 2 \pmod{T_H}$. So $p < f(H)$ for some function f by Dirichlet's Theorem (we don't need any specific estimates here). Since $p-1 \equiv 1 \pmod{T_H}$, we have $\gcd(p-1, T_H) = 1$ and therefore $\gcd(p-1, pT_H) = 1$. Now, by the Chinese Remainder Theorem, select the smallest positive integer k' satisfying $k' \equiv n \pmod{pT_H}$ and $k' \equiv 1 \pmod{p-1}$. Note that $k' < p^2 T_H$. Let $k = (n-k')/(pT_H)$; so we have $n = kpT_H + k'$.

Let $N' = \binom{n}{2} - \binom{k'}{2}$. Consider the following Boolean function B_n^H on N' variables. Consider graphs X on the vertex set $[n]$ with the property that they have no edges among their last k' vertices. These graphs can be viewed as Boolean functions of the remaining N' variables. Now we say that such a graph has property B_n^H if it does not contain H as a subgraph.

Claim. The function B_n^H is evasive.

The Claim immediately implies that the query complexity of Q_n^H is at least N', proving the Theorem with $C_H = \binom{k'}{2} < p^4 T_H^2 < f(H)^4 T_H^2$.

To prove the Claim, consider the groups $\Lambda := (\mathbb{F}_p^+)^{kT_H} \rtimes \mathbb{F}_p^\times$ and $\Gamma := \Lambda \times \mathbb{Z}_{k'}$. Here Λ acts on $[pkT_H]$ in the obvious way: we divide $[pkT_H]$ into kT_H blocks of size p; $\mathbb{F}_p^+$ acts on each block independently and $\mathbb{F}_p^\times$ acts on the blocks simultaneously (diagonal action) so on each block they combine to an $\mathrm{AGL}(1,p)$-action. $\mathbb{Z}_{k'}$ acts as a k'-cycle on the remaining k' vertices. So Γ is a cyclic extension of a p-group (because $\gcd(p-1,k') = 1$).

If B_n^H is not evasive then from Theorem 2.1 and Lemma 2.2, we have $\chi\left((B_n^H)_\Gamma\right) = 1$.

On the other hand we claim that, $(B_n^H)_\Gamma \cong Q_r^{[[H]]}$, where $r = kT_H + 1$. The proof of this claim is exactly the same as the proof of Lemma 4.2 of [3]. Thus, from Lemma 3.1, we conclude that $\chi(Q_r^{[[H]]})$ is even. This contradicts the previous conclusion that $\chi(Q_r^{[[H]]}) = 1$. ■

Remark 3.8. Specific estimates on the smallest Dirichlet prime can be used to estimate C_H. Linnik's theorem implies $C_H < h^{O(1)}$, extending Theorem 3.7 to strong lower bounds for variable H up to $h = n^c$ for some positive constant c.

4. Sparse graphs: unconditional results

We prove part (c) of Theorem 1.4.

Theorem 4.1. *If the non-empty monotone graph property P_n is not evasive then*

$$\dim(P_n) = \Omega(n \log n).$$

4.1. The basic group construction

Assume in this section that $n = p^\alpha k$ where p is prime. Let $\Delta_k \leq \Sigma_k$. We construct the group $\Gamma_0(p^\alpha, \Delta_k)$ acting on $[n]$.

Let $\Delta = (\mathbb{F}_{p^\alpha}^\times \times \Delta_k)$. Let $\Gamma_0(p^\alpha, \Delta_k)$ be the semidirect product $(\mathbb{F}_{p^\alpha}^+)^k \rtimes \Delta$ with respect to the Δ-action on $(\mathbb{F}_{p^\alpha}^+)^k$ defined by

$$(a, \sigma) : (b_1, \ldots, b_k) \mapsto (ab_{\sigma^{-1}(1)}, \ldots, ab_{\sigma^{-1}(k)}).$$

We describe the action of $\Gamma_0(p^\alpha, \Delta_k)$ on $[n]$. Partition $[n]$ into k clusters of size p^α each. Identify each cluster with the field of order p^α, i.e., as a set, $[n] = [k] \times \mathbb{F}_{p^\alpha}$. The action of $\gamma = (b_1, \ldots, b_k, a, \sigma)$ is described by

$$\gamma : (x, y) \mapsto (\sigma(x), ay + b_{\sigma(x)}).$$

An unordered pair $(i, j) \in [n]$ is termed an *intra-cluster edge* if both i and j are in the same cluster, otherwise it is termed an *inter-cluster edge.* Note that every u-orbital under Γ has only intra-cluster edges or only inter-cluster edges. Denote by m_{intra} and m_{inter} the minimum sizes of u-orbitals of intra-cluster and inter-cluster edges respectively.

We denote by m'_k the minimum size of an orbit in $[k]$ under Δ_k and by m''_k the minimum size of a u-orbital in $[k]$. We then have:

$$m_{\text{intra}} \geq \binom{p^\alpha}{2} \times m'_k, \qquad m_{\text{inter}} \geq (p^\alpha)^2 \times m''_k$$

Let m_k‘ $:= \min\{m'_k, m''_k\}$ and define m^* as the minimum size of a u-orbital in $[n]$. Then

$$m^* = \min\{m_{\text{intra}}, m_{\text{inter}}\} = \Omega(p^{2\alpha} m_k) \tag{4.1}$$

4.2. Vinogradov's Theorem

The Goldbach Conjecture asserts that every even integer can be written as the sum of two primes. Vinogradov's Theorem [24] says that every sufficiently large odd integer k is the sum of three primes $k = p_1 + p_2 + p_3$. We use here Haselgrove's version [7] of Vinogradov's theorem which states that we can require the primes to be roughly equal: $p_i \sim k/3$. This can be combined with the Prime Number Theorem to conclude that every sufficiently large even integer k is a sum of four roughly equal primes.

4.3. Construction of the group

Let $n = p^\alpha k$ where p is prime. Assume k is not bounded. Write k as a sum of $t \leq 4$ roughly equal primes p_i. Let $\Delta_k := \prod_i \mathbb{Z}_{p_i}$ where $\mathbb{Z}_{p_i}$ denotes the cyclic group of order p_i and the direct product is taken over the *distinct* p_i.

Δ_k acts on $[k]$ as follows: partition k into parts of sizes $p_1, \ldots, p_t$ and call these parts $[p_i]$. The group $\mathbb{Z}_{p_i}$ acts as a cyclic group on the part $[p_i]$. In case of repetitions, the same factor $\mathbb{Z}_{p_i}$ acts on all the parts of size p_i.

We follow the notation of Section 4.1 and consider the group $\Gamma_0(p^\alpha, \Delta_k)$ with our specific Δ_k. We have $m_k = \Omega(k)$ and hence we get, from equation (4.1):

Lemma 4.2. *Let $n = p^\alpha k$ where p is a prime. For the group $\Gamma_0(p^\alpha, \Delta_k)$, we have $m^* = \Omega(p^{2\alpha} k) = \Omega(p^\alpha n)$, where m^* denotes the minimum size of a u-orbital.*

4.4. Proof for the superlinear bound

Let $n = p^\alpha k$ where p^α is the largest prime power dividing n; so $p^\alpha = \Omega(\log n)$; this will be a lower bound on the size of u-orbitals. Our group Γ will be of the general form discussed in Section 4.1.

Case 1. $p^\alpha = \Omega(n^{2/3})$.
Let $\Gamma = \Gamma_0(p^\alpha, \{1\})$. Following the notation of Section 4.1, we get $m'_k = m''_k = 1$, and this yields that $m^* = \Omega((p^\alpha)^2) = \Omega(n^{4/3}) = \Omega(n \log n)$. Oliver's condition is easily verified for Γ.

Case 2. $k = \Omega(n^{1/3})$.
Consider the $\Gamma := \Gamma_0(p^\alpha, \Delta_k)$ acting on $[n]$ where Δ_k is as described in Section 4.3. The minimum possible size m^* of a u-orbital is $\Omega(np^\alpha)$ by Lemma 4.2. Finally, since $p^\alpha = \Omega(\log n)$, we obtain $m^* = \Omega(n \log n)$.

If all p_i are co-prime to $p^\alpha - 1$ then $\mathbb{F}_{p^\alpha}^\times \times \Delta_k$ becomes a cyclic group and Γ becomes a cyclic extension of a p-group.

Since $p_i = \Omega(k) = \Omega(n^{1/3})$ for all i and $p^\alpha = O(n^{2/3})$, size considerations yield that at most one p_i divides $p^\alpha - 1$ and p_i^2 does not. Suppose, without loss of generality, p_1 divides $p^\alpha - 1$. Let $p^\alpha - 1 = p_1 d$, then d must be co-prime to each p_i. Thus, $\Delta = (\mathbb{Z}_{p_1} \times \mathbb{Z}_d) \times (\mathbb{Z}_{p_1} \times \ldots \times \mathbb{Z}_{p_t}) = (\mathbb{Z}_d \times \mathbb{Z}_{p_2} \times \ldots \times \mathbb{Z}_{p_r}) \times (\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_1})$. Thus, Δ is a p_1-group extension of a cyclic group. Hence, Γ satisfies Oliver's Condition (cf. Theorem 2.1). ■

Remark 4.3. For almost all n, our proof gives a better dimension lower bound of $\Omega(n^{1+\frac{1+o(1)}{\ln \ln n}})$.

5. Sparse graphs: conditional improvements

In this section we prove parts (a) and (b) of Theorem 1.4.

5.1. General Setup

Let $n = pk + r$, where p and r are prime numbers. Let q be a prime divisor of $(r-1)$. We partition $[n]$ into two parts of size pk and r, denoted by $[pk]$ and $[r]$ respectively. We now construct a group $\Gamma(p, q, r)$ acting on $[n]$ as a direct product of a group acting on $[pk]$ and a group acting on $[r]$, as follows:

$$\Gamma = \Gamma(p, q, r) := \Gamma_0(p, \Delta_k) \times \Gamma(r, q)$$

Here, $\Gamma_0(p, \Delta_k)$ acts on $[pk]$ and is as defined in Section 4.3, and involves choosing a partition of k into upto four primes that are all $\Omega(k)$.

$\Gamma(r, q)$ is defined as the semidirect product $\mathbb{F}_r^+ \rtimes C_q$, with C_q viewed as a subgroup of the group $\mathbb{F}_r^\times$. It acts on $[r]$ as follows: We identify $[r]$ with the field of size r. Let (b, a) be a typical element of Γ_r where $b \in \mathbb{F}_r$ and $a \in C_q$. Then, $(b, a) : x \mapsto ax + b$.

Thus, $\Gamma = \Gamma(p, q, r)$ acts on $[n]$. Let m^* be the minimum size of the orbit of any edge $(i, j) \in \binom{[n]}{2}$ under the action of Γ. One can show that

$$m^* = \Omega(\min\{p^2 k, pkr, qr\}). \tag{5.1}$$

We shall choose p, q, r carefully such that (a) the value of m^* is large, and (b) Oliver's condition holds for $\Gamma(p, q, r)$.

5.2. ERH and Dirichlet primes

The Extended Riemann Hypothesis (ERH) implies the following strong version of the Prime Number Theorem for arithmetic progressions. Let $\pi(n, D, a)$ denote the numer of primes $p \leq n$, $p \equiv a \pmod{D}$. Then for $D < n$ we have

$$\pi(n, D, a) = \frac{\mathrm{li}(n)}{\varphi(D)} + O(\sqrt{x} \ln x) \tag{5.2}$$

where $\mathrm{li}(n) = \int_2^n dt/t$ and the constant implied by the big-Oh notation is absolute (cf. [16, Ch. 7, eqn. (5.12)] or [2, Thm. 8.4.5]).

This result immediately implies "Bertrand's Postulate for Dirichlet primes:"

Lemma 5.1 (Bertrand's Postulate for Dirichlet primes)**.** *Assume ERH. Suppose the sequence D_n satisfies $D_n = o(\sqrt{n}/\log^2 n)$. Then for all sufficiently large n and for any a_n relatively prime to D_n there exists a prime $p \equiv a_n \pmod{D_n}$ such that $\frac{n}{2} \leq p \leq n$.*

5.3. With ERH but without Chowla

We want to write $n = pk + r$, where p and r are primes, and with q a prime divisor of $r - 1$, as described in Section 5.1. Specifically, we try for:

$$p = \Theta(n^{1/4}), \quad \frac{n}{4} \leq r \leq \frac{n}{2}, \quad q = \Theta(n^{1/4-\epsilon})$$

We claim that under ERH, such a partition of n is possible.

To see this, fix some $p = \Theta(n^{1/4})$ such that $\gcd(p, n) = 1$. Fix some $q = \Theta(n^{1/4-\epsilon})$. Now, $r \equiv 1 \pmod{q}$ and $r \equiv n \pmod{p}$ solves to $r \equiv a \pmod{pq}$ for some a such that $\gcd(a, pq) = 1$. Since $pq = \Theta(n^{1/2-\epsilon})$, we can conclude under ERH (using Lemma 5.1) that there exists a prime $r \equiv a \pmod{pq}$ such that $\frac{n}{4} \leq r \leq \frac{n}{2}$. This gives us the desired partition. One can verify that our Γ satisfies Oliver's Condition. Equation (5.1) gives $m^* = \Omega(n^{5/4-\epsilon})$. This completes the proof of part (b) of Theorem 1.4. ■

5.4. Stronger bound using Chowla's conjecture

Let a and D be relatively prime. Let p be the first prime such that $p \equiv a \pmod{D}$. Chowla's conjecture tells us that $p = O(D^{1+\epsilon})$ for every $\epsilon > 0$. Using this, we show $m^* = \Omega(n^{3/2-\epsilon})$.

We can use Chowla's conjecture, along with the general setup of Section 5.1, to obtain a stronger lower bound on m^*. The new bounds we hope to achieve are:

$$p = \Theta(\sqrt{n}), \quad n^{1-2.5\delta} \leq r \leq n^{1-0.5\delta}, \quad q = \Theta(n^{1/2-\delta})$$

Such a partition is always possible assuming Chowla's conjecture. To see this, first fix $p = \Theta(n^{1/2})$, then fix $q = \Theta(n^{1/2-2\delta})$ and find the least solution for $r \equiv 1 \pmod{q}$ and $r \equiv n \pmod{p}$, which is equivalent to solving for $r \equiv a \pmod{pq}$ for some $a < pq$. The least solution will be greater than pq unless a happens to be a prime. In this case, we add another constraint, say $r \equiv a + 1 \pmod{3}$ and resolve to get the least solution greater than pq. Note that $n^{1-2.5\delta} \leq r \leq n^{1-0.5\delta}$. Now, from Equation (5.1), we get the lower bound of $m^* = \Omega(n^{3/2-4\delta})$. This completes the proof of part (a) of Theorem 1.4. ■

Acknowledgment.

Raghav Kulkarni expresses his gratitude to Sasha Razborov for bringing the subject to his attention and for helpful initial discussions.

References

[1] Babai, L., Gál, A., Wigderson, A.: Superpolynomial lower bounds for monotone span programs. *Combinatorica* 19 (1999), 301–320.

[2] Bach, E., Shallit, J.: *Algorithmic Number Theory, Vol. 1.* The MIT Press 1996.

[3] Chakrabarti, A., Khot, S., Shi, Y.: Evasiveness of Subgraph Containment and Related Properties. *SIAM J. Comput.* 31(3) (2001), 866-875.

[4] Chowla, S. On the least prime in the arithmetical progression. *J. Indian Math. Soc.* 1(2) (1934), 1–3.

[5] Davenport, H.: *Multiplicative Number Theory.* (2nd Edn) Springer Verlag, New York, 1980.

[6] Granville, A., Pomerance, C.: On the least prime in certain arithmetic progressions. *J. London Math. Soc.* 41(2) (1990), 193–200.

[7] Haselgrove, C. B.: Some theorems on the analytic theory of numbers. *J. London Math. Soc.* 36 (1951) 273–277

[8] Heath-Brown, D. R.: Almost-primes in arithmetic progressions and short intervals. *Math. Proc. Cambr. Phil. Soc.* 83 (1978) 357–376.

[9] Heath-Brown, D. R.: Zero-free regions for Dirichlet L-functions, and the least prime in an arithmetic progression. *Proc. London Math. Soc.* 64(3) (1992) 265–338.

[10] Kleitman, D. J., Kwiatkowski, D. J.: Further results on the Aanderaa-Rosenberg Conjecture *J. Comb. Th. B* 28 (1980), 85–90.

[11] Kahn, J., Saks, M., Sturtevant, D.: A topological approach to evasiveness. *Combinatorica* 4 (1984), 297–306.

[12] Lutz, F. H.: Examples of $\mathbb{Z}$-acyclic and contractible vertex-homogeneous simplicial complexes.. *Discrete Comput. Geom.* 27 (2002), No. 1, 137–154.

[13] Mader, W.: Homomorphieeigenschaften und mittlere Kantendichte von Graphen. *Math. Ann.* 174 (1967), 265–268.

[14] Mader, W.: Homomorphiesätze für Graphen. *Math. Ann.* **175** (1968), 154–168.

[15] Oliver, R.: Fixed-point sets of group actions on finite acyclic complexes. *Comment. Math. Helv.* 50 (1975), 155–177.

[16] Prachar, K.: *Primzahlverteilung.* Springer, 1957.

[17] Rosenberg A. L.: On the time required to recognize properties of graphs: A problem. *SIGACT News* 5 (4) (1973), 15–16.

[18] Rotman, J.: *An Introduction to the Theory of Groups.* Springer Verlag, 1994.

[19] Rivest, R.L., Vuillemin, J.: On recognizing graph properties from adjacency matrices. *Theoret. Comp. Sci.* 3 (1976), 371–384.

[20] Smith P. A.: Fixed point theorems for periodic transformations. *Amer. J. of Math.* 63 (1941), 1–8.

[21] Titchmarsh, E. C.: A divisor problem. *Rend. Circ. Mat. Palermo* 54 (1930), 419–429.

[22] Triesch, E.: On the recognition complexity of some graph properties. *Combinatorica* 16 (2) (1996) 259–268.

[23] Turán, P.: Über die Primzahlen der arithmetischen Progression. *Acta Sci. Math. (Szeged)* 8 (1936/37) 226–235.

[24] Vinogradov, I. M.: *The Method of Trigonometrical Sums in the Theory of Numbers (Russian).* Trav. Inst. Math. Stekloff 10, 1937.

[25] Yao, A. C.: Monotone bipartite properties are evasive. *SIAM J. Comput.* 17 (1988), 517–520.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 83-94
www.stacs-conf.org

DYNAMIC SHARING OF A MULTIPLE ACCESS CHANNEL

MARCIN BIENKOWSKI[1] AND MAREK KLONOWSKI[2] AND MIROSLAW KORZENIOWSKI[2] AND DARIUSZ R. KOWALSKI[3]

[1] Institute of Computer Science, University of Wrocław, Poland

[2] Institute of Mathematics and Computer Science, Wrocław University of Technology, Poland

[3] Department of Computer Science, University of Liverpool, UK

In this paper we consider the mutual exclusion problem on a multiple access channel. Mutual exclusion is one of the fundamental problems in distributed computing. In the classic version of this problem, n processes perform a concurrent program which occasionally triggers some of them to use shared resources, such as memory, communication channel, device, etc. The goal is to design a distributed algorithm to control entries and exits to/from the shared resource in such a way that in any time there is at most one process accessing it. We consider both the classic and a slightly weaker version of mutual exclusion, called ε-mutual-exclusion, where for each period of a process staying in the critical section the probability that there is some other process in the critical section is at most ε. We show that there are channel settings, where the classic mutual exclusion is not feasible even for randomized algorithms, while ε-mutual-exclusion is. In more relaxed channel settings, we prove an exponential gap between the makespan complexity of the classic mutual exclusion problem and its weaker ε-exclusion version. We also show how to guarantee fairness of mutual exclusion algorithms, i.e., that each process that wants to enter the critical section will eventually succeed.

1. Introduction

In this paper we consider randomized algorithms for mutual exclusion: one of the fundamental problems in distributed computing. We assume that there are n different processes labeled from 0 to $n-1$ communicating through a multiple access channel (MAC). The computation and communication proceed in synchronous slots, also called *rounds*. In the mutual exclusion problem, each process performs a concurrent program and occasionally requires exclusive access to shared resources. The part of the code corresponding to this exclusive access is called a *critical section*. The goal is to provide a mechanism that controls

1998 ACM Subject Classification: C.1.4 Parallel Architectures, C.2.1 Network Architecture and Design, F.2.2 Nonnumerical Algorithms and Problems. Supported by Polish Ministry of Science and Higher Education grants no N N206 2573 35 and N N206 1723 33, and by the Engineering and Physical Sciences Research Council [grant number EP/G023018/1].

Key words and phrases: distributed algorithms, multiple access channel, mutual exclusion.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2446

entering and exiting the critical section and guarantees exclusive access at any time. The main challenge is that the designed mechanism must be universal, in the sense that exclusive access must be guaranteed regardless of the times of access requests made by other processes.

Multiple Access Channel (MAC). We consider a multiple access channel as both communication medium and the shared-access device. As a communication medium, MAC allows each process either to transmit or listen to the channel at a round,[1] and moreover, if more than one process transmits, then a *collision* (signal interference) takes place. Depending on the devices used in the system, there are several additional settings of MAC that need to be considered. One of them is the ability of a process to distinguish between background noise when no process transmits (also called *silence*) and collision. If such capability is present at each process, we call the model *with collision detection* (CD for short); if no process has such ability, then we call the setting *without collision detection* (no-CD). Another feature of the model is a constant access to the global clock (GC for short) by all processes or no such access by any of them (no-GC). The third parameter to be considered is a knowledge of the total number of available processes n (KN for short) or the lack of it (no-KN).

Mutual Exclusion Problem. In this problem, each concurrent process executes a protocol partitioned into the following four sections:

Entry: the part of the protocol executed in preparation for entering the critical section;
Critical: the part of the protocol to be protected from concurrent execution;
Exit: the part of the protocol executed on leaving the critical section;
Remainder: the rest of the protocol.

These sections are executed cyclically in the order: *remainder, entry, critical, and exit.* Intuitively, the remainder section corresponds to local computation of a process, and the critical section corresponds to the access to the shared object (the channel in our case); though the particular purpose and operations done within each of these sections are not a part of the problem. Sections entry and exit are the parts that control switching between remainder and critical sections in a process, in order to assure some desired properties of the whole system.

In the traditional mutual exclusion problem, as defined in [1, 16] in the context of shared-memory model, the adversary controls the sections remainder and critical. In particular, she controls their duration in each cycle, subject only to the obvious assumptions that this duration in each cycle is finite or the last performed section is the remainder one. The mutual exclusion algorithm, on the other hand, provides a protocol for the entry and exit sections of each process. In this sense, the mutual exclusion problem can be seen as a game between the adversary controlling the lengths of remainder and critical sections of each process (each such section for each process may have different length) and the algorithm controlling entry and exit sections. The goal of the algorithm is to guarantee several useful properties of the execution (to be defined later), while the goal of the adversary is to prevent it. Note that the sections controlled by the adversary and those controlled by the algorithm are interleaved in the execution. Additionally, in order to make the game fair, it is typically

[1] Most of the previous work on MAC, motivated by Ethernet applications, assumed that a process can transmit and listen simultaneously; our work instead follows the recent trends of wireless applications where such simultaneous activities are excluded due to physical constraints.

assumed that every variable used by the algorithm, i.e., in the entry and exit sections, cannot be modified by the adversary in the critical and remainder sections, and vice versa, i.e., no variables used by the adversary in the remainder and critical sections can be accessed by the algorithm.

In the model of communication over MAC, a process in the entry or the exit section can do the following in a single round: perform some action on the channel (either transmit a message or listen), do some local computation, and change its section either from entry to critical or from exit to remainder. We assume that changing sections occurs momentarily between consecutive rounds, i.e., in each round a process is exactly in one section of the protocol.

Since a multiple-access channel is both the only communication medium and the exclusively shared object, additional constraints, different from the classic ones regarding e.g., shared memory objects, must be imposed:

- no process in the remainder section is allowed to transmit on the channel;
- a process in the critical section has to transmit a message on the channel each round until it moves to the exit section, and each such message must be labelled *critical*; we call them *critical messages*.

If any of these conditions was violated, the adversary would have an unlimited power of creating collisions on the channel, and thus preventing any communication.

A classic mutual exclusion algorithm should satisfy the following three properties for any round i of its execution:

Exclusion: at most one process is in the *critical* section in round i.

Unobstructed exit: if a process p is in the exit section at round i, then process p will switch to the remainder section eventually after round i.

No deadlock: if there is a process in the entry section at round i, then *some* process will enter the critical section eventually after round i.

To strengthen the quality of service guaranteed by mutual exclusion algorithms, the following property — stronger than no-deadlock — has been considered:

No lockout: if a process p is in the entry section at round i, then *process* p will enter the critical section eventually after round i.

Note that — to some extent — this property ensures fairness: each process demanding an access to the critical section will eventually get it.

As we show, in some cases the exclusion condition is impossible or very costly to achieve. Therefore, we also consider a slightly weaker condition:

ε-exclusion: for every process p and for every time interval in which p is continuously in the critical section, the probability that in any round of this time interval there is another process being in the critical section is at most ε.

Intuitively, ε-exclusion guarantees mutual exclusion “locally”, i.e., for every single execution of the critical section by a process, with probability at least $1-\varepsilon$. The version of the problem satisfying ε-exclusion condition is called *ε-mutual-exclusion.*

Complexity Measure. We use the *makespan* measure, as defined in [8] in the context of deterministic algorithms. Makespan of an execution of a given deterministic mutual exclusion algorithm is defined as the maximum length of a time interval in which there is some process in the entry section and there is no process in the critical section. Taking maximum of such values over all possible executions defines the makespan of the algorithm.

In order to define expected makespan, suitable for randomized algorithms considered in this work, we need more formal definitions of an adversarial strategy. Let $\mathcal{P}$ be a strategy of the adversary, defined as a set of n sequences, where each sequence corresponds to a different process and contains, subsequently interleaved, lengths of remainder and critical sections of the corresponding process. We assume that each sequence is either infinite or of even length; the latter condition means that after the last critical section the corresponding process runs the remainder section forever. For a given mutual exclusion algorithm $\textsc{Alg}$ and adversarial strategy $\mathcal{P}$, we define $L(\textsc{Alg}, \mathcal{P})$ as a random variable equal to the maximum length of a time interval in which there is some process in the entry section and there is no process in the critical section in an execution of $\textsc{Alg}$ run against fixed strategy $\mathcal{P}$. The expected makespan of algorithm $\textsc{Alg}$ is defined as the maximum of expected values of $L(\textsc{Alg}, \mathcal{P})$, taken over all adversarial strategies $\mathcal{P}$. Note that every algorithm with makespan bounded for all executions satisfies no-deadlock property, but not necessarily no-lockout.

For the ε-mutual-exclusion problem, defining makespan is a bit more subtle. We call an execution *admissible* if the mutual exclusion property is always fulfilled, i.e., no two processes are in the critical section in the same round. Then in the computation of the (expected) makespan, we neglect non-admissible executions.

1.1. Our Results

We consider the mutual exclusion problem and its weaker ε-exclusion version in the multiple access channel. Unlike the previous paper [8], where only no-deadlock property was guaranteed, we also focus on fairness. Also in contrast to the previous work on mutual exclusion on MAC, we mostly study randomized solutions. In the case of the mutual exclusion problem, we allow randomized algorithms to have variable execution time but they have to be always correct. On the other hand, a randomized solution for the ε-mutual-exclusion problem is allowed to err with some small probability ε. Thus, for the former problem, we require Las Vegas type of solution, whereas for the latter we admit Monte Carlo algorithms. Note that very small (e.g., comparable with probability of hardware failure) risk of failure (i.e., situation wherein two or more processes are in the critical section at the same round) is negligible from a practical point of view.

We show that for the most severe channel setting, i.e., no-CD, no-GC and no-KN, mutual exclusion is not feasible even for randomized algorithms (cf. Section 2).

In a more relaxed setting, there is an exponential gap between the complexity of the mutual exclusion problem and the ε-mutual-exclusion problem. Concretely, we prove that the expected makespan of (randomized) solutions for the mutual exclusion problem in the no-CD setting is $\Omega(n)$, even if the algorithm knows n, has access to the global clock (cf. Section 2), and even if only no-deadlock property is required. On the other hand, for the ε-mutual-exclusion problem, we construct a randomized algorithm, requiring only the knowledge of n, which guarantees no-lockout property, and whose makespan is $O(\log n \cdot \log(1/\varepsilon))$ (cf. Sections 3.2 and 4).

When collision detection is available and only no-deadlock property is required, we show that the makespan of any mutual exclusion algorithm is at least $\Omega(\log n)$ (cf. Section 2) and we construct an algorithm for the ε-mutual-exclusion problem with expected makespan $O(\log \log n + \log(1/\varepsilon))$ (cf. Section 3.3). Further, we show how to modify this algorithm to guarantee no-lockout property as well; its expected makespan becomes then $O(\log n + \log(1/\varepsilon))$ (cf. Sections 3.3 and 4).

Finally, if we do not require no-lockout property, we show how to solve the ε-mutual-exclusion problem in makespan $O(\log n \cdot \log(1/\varepsilon))$, where only the global clock is available (cf. Section 3.1).

We also present a generic method that, taking a mutual exclusion algorithm with no-deadlock property, turns it into the one satisfying stronger no-lockout condition. This method applied to the deterministic algorithms from [8] produces efficient deterministic solutions satisfying the no-lockout property.

Due to space limitations, the missing details and proofs will appear in the full version of the paper.

1.2. Previous and Related Work

The multiple access channel is a well-studied model of communication. In many problems considered in this setting, one of the most important issues is to assure that successful transmissions occur in the computation. These problems are often called *selection problems*. They differ from the mutual exclusion problem by the fact that they focus on successful transmissions within a bounded length period, while mutual exclusion provides control mechanism for dynamic and possibly unbounded computation. In particular, it includes recovering from long periods of cumulative requests for the critical section as well as from long periods containing no request. Additionally, selection problems were considered typically in the context of Ethernet or combinatorial group testing, and as such they allowed a process to transmit and to listen simultaneously, which is not the case in our model motivated by wireless applications. Selection problems can be further split into two categories. In the static selection problems, it is assumed that a subset of processes become active at the same time and a subset of them must eventually transmit successfully. Several scenarios and model settings, including parameters considered in this work such as CD/no-CD, GC/no-GC, KN/no-KN, randomization/determinism, were considered in this context, see e.g., [2, 4, 7, 11, 12, 14, 15, 17, 18, 19]. In the *wake-up* problem, processes are awaken in (possibly) different rounds and the goal is to assure that there will be a round with successful transmission ("awakening" the whole channel) shortly after the first process is awaken, see, e.g., [5, 9, 13].

More dynamic kinds of problems, such as transmission of dynamically arriving packets, were also considered in the context of MAC. In the (dynamic) packet transmission problem, the aim is to obtain bounded throughput and bounded latency. Two models of packet arrival were considered: stochastic (cf., [10]) and adversarial queuing (cf., [3, 6]). There are two substantial differences between these settings and our work. First, the adversaries imposing dynamic packet arrival are different than the adversary simulating execution of concurrent protocol. Second, as already mentioned in the context of selection problems, these papers were inspired by Ethernet applications where it is typically allowed to transmit and listen simultaneously.

In a very recent paper [8] *deterministic* algorithms for mutual exclusion problem in MAC under different settings (CD, GC, KN) were studied. The authors proved that with none of those three characteristics mutual exclusion is infeasible. Moreover, they presented an optimal — in terms of the makespan measure — $O(\log n)$ round algorithm for the model with CD. They also developed algorithms achieving makespan $O(n \log^2 n)$ in the models with GC or KN only, which, in view of the lower bound $\Omega(n)$ on deterministic solutions proved for any model with no-CD, is close to optimal. Our paper differs from [8] in three ways. First, we consider both deterministic and randomized solutions. Second, for the sake of efficiency we introduce the ε-mutual-exclusion problem. Third, we study fairness of protocols, which means that we consider also no-lockout property.

2. Lower Bounds for the Mutual Exclusion Problem

In our lower bounds, we use the concept of transmission schedules to capture transmission/listening activity of processes in the entry or exit section. Transmission schedule of a process p can be regarded as a binary sequence π_p describing the subsequent communication actions of the process. The sequence can be finite or infinite. For non-negative integer i, $\pi_p(i) = 1$ means that process p transmits in round i after starting its current section, while $\pi_p(i) = 0$ means that the process listens in round i. We assume that round 0 is the round in which the process starts its current run of the entry or the exit section.

The following results extend the lower bounds and impossibility results for deterministic mutual exclusion proved in [8] to randomized solutions. All the presented lower bounds work even if we do not require no-lockout, but a weaker no-deadlock property.

Theorem 2.1. *There is no randomized mutual exclusion algorithm with no-deadlock property holding with a positive probability in the setting without collision detection, without global clock and without knowledge of the number n of processes.*

Theorem 2.2. *The expected makespan of any randomized mutual exclusion algorithm is at least $\log n$, even in the setting with collision detection, with global clock and with knowledge of the number n of processes.*

Theorem 2.3. *The expected makespan of any randomized mutual exclusion algorithm is at least $n/2$ in the absence of collision detection capability, even in the setting with global clock and with knowledge of the number n of processes.*

Proof. To arrive at a contradiction, let $\mathcal{R}$ be a randomized mutual exclusion algorithm, whose expected makespan is c, where $c < n/2$. We show that there exists an execution violating mutual exclusion.

Let $\mathcal{E}_p^*$, for process p, be the set of all possible executions of the first entry section of algorithm $\mathcal{R}$ by process p under the assumption that it starts its first entry section in the global round 1 and there is no other process starting within the first $n/2$ rounds. Note that during each execution in $\mathcal{E}_p^*$ process p hears only noise (i.e., silence or collision, which are indistinguishable due to the lack of collision detection) from the channel when listening. Observe also that the optimum algorithm needs only one round to let process p enter the critical section under the considered adversarial scenario. Therefore, by the probabilistic method, there is an execution $\mathcal{E}_p$ in set $\mathcal{E}_p^*$ where process p enters the critical section within the first $n/2 - 1$ rounds. Let π_p be the transmission schedule of process p during $\mathcal{E}_p$.

Consider all sequences π_p over all processes $0 \le p < n$. We construct execution $\mathcal{E}$ contradicting mutual exclusion as follows. First, we need to select a set of processes that start their first entry sections in round 1, while the others stay in the remainder section till at least round $n/2$. Let $P_0 = \{0, \ldots, n-1\}$. For every non-negative integer j, we define recursively

$$
\begin{aligned}
P_{2j+1} &= P_{2j} \setminus \left\{p \in P_{2j} : \exists_{i\in[1,n/2-1]} \left(\pi_p(i) = 1 \ \& \ \forall_{q\in P_{2j}, q\neq p}\ \pi_q(i) = 0\right)\right\} \ , \\
P_{2j+2} &= P_{2j+1} \setminus \left\{p \in P_{2j+1} : \exists_{i\in[1,n/2-1]} \left(|\pi_p| = i \ \& \ \forall_{q\in P_{2j+1}}\ |\pi_q| > i\right)\right\} \ .
\end{aligned}
$$

Intuitively, set P_{2j+1} is obtained from P_{2j} by removing processes p that could be single transmitters in some round in the interval $[1, n/2 - 1]$ while transmitting according to their schedules π_p. Set P_{2j+2} is constructed by removing a process with the shortest transmission schedule, if there is only one such process. Observe that sequence $\{P_j\}_{j\geq 0}$ is bounded and monotonically non-increasing (in the sense of set inclusion), therefore it stabilizes on some set P^*. Observe that

(1) $|P^*| \geq 2$, since for each round $i \in [1, n/2 - 1]$ there is at most one process removed from some set P_{2j} while constructing the consecutive set P_{2j+1} (after such removal no remaining process has 1 in position i of its schedule) and at most one process removed from some set $P_{2j'+1}$ while constructing the consecutive set $P_{2j'+2}$ (after such removal no remaining process p finishes its transmission schedule π_p in round i); as there are $n/2 - 1$ considered rounds, at most $n - 2$ processes can be removed throughout the construction;

(2) there is no round $i \in [1, n/2-1]$ such that there is only one process $p \in P^*$ satisfying $\pi_p(i) = 1$; this follows from the fact that P^* is a fixed point of the sequence $\{P_j\}_{j\geq 0}$, i.e., it does not change while applying the odd-step rule of the construction;

(3) there are at least two processes $p, q \in P^*$ with the shortest transmission schedules π_p, π_q, i.e., $|\pi_p| = |\pi_q|$ and for every process $r \in P^*$, $|\pi_r| \geq |\pi_p|$; this again follows from the fact that P^* is a fixed point of the sequence $\{P_j\}_{j\geq 0}$, i.e., it does not change while applying the even-step rule of the construction.

Having subset P^* of processes, the adversary starts first entry sections for all processes in P^* in the very first round, while she delays others (they remain in the remainder section) by round $n/2$. Note that before round 1 of the constructed execution $\mathcal{E}$, a process $p \in P^*$ cannot distinguish $\mathcal{E}$ from $\mathcal{E}_p$, therefore it may decide to do the same as in $\mathcal{E}_p$, i.e., to set its first position of transmission schedule to $\pi_p(1)$. If this happens for all processes in P^*, by the second property of this set there is no single transmitter in round 1, and therefore all listening processes hear the noise (recall that silence is not distinguishable from collision in the considered setting). This construction and the output of the first round can be inductively extended up to round $|\pi_p|$, where $p \in P^*$ is a process with the shortest schedule π_p among processes in P^*. This is because from the point of view of a process $q \in P^*$ the previously constructed prefix of $\mathcal{E}$ is not distinguishable from the corresponding prefix of execution $\mathcal{E}_q$; indeed, the transmission schedules are the same and the feedback from the channel is silence whenever the process listens. Finally, by the very same reason, at the end of round $|\pi_p|$ all processes $q \in P^*$ with $|\pi_q| = |\pi_p|$ are allowed to do in $\mathcal{E}$ the same action as in $\mathcal{E}_q$, that is, to enter the critical section. By the third property of set P^*, there is at least one such process $q \in P^*$ different than p. This violates the exclusion property that should hold for the constructed execution $\mathcal{E}$. ■

3. Algorithms for the ε-Mutual-Exclusion Problem

In this section, we present randomized algorithms solving the ε-mutual-exclusion problem for various scenarios, differing in the channel capabilities (e.g., CD/no-CD, KN/no-KN, GC/no-GC). The algorithms presented in this section, work solely in entry sections, i.e., their exit sections are empty; these algorithms guarantee only no-deadlock property. However, in Section 4, we show how to add exit section subroutines to most of our algorithms to guarantee the no-lockout property while keeping bounded makespan. In our algorithms, we extend some techniques developed in the context of other related problems, such as the wake-up problem [13] and the leader election problem [19].

Throughout this section, we use the following notation. We say that there is a *successful transmission* in a given round if in this round one process transmits and other processes do not transmit. By saying that a process *resigns*, we mean that it will not try to enter the critical section and will not attempt to transmit anything until another process starts the exit section.

3.1. Only Global Clock Available

In the model with global clock, we modify the *Increase_From_Square* algorithm [13], which solves the wake-up problem. The purpose of our modification is to assure the stopping property. This is a nontrivial task in a scenario without collision detection and this property was not present in the original wake-up algorithm. Intuitively, after one process successfully transmits, it should enter the critical section. However, first of all it might not be aware that it succeeded. Second, between a successful transmission and entering the critical section, some other processes may start their entry sections. The details will be presented in the full version of this paper.

Theorem 3.1. *There is an ε-mutual-exclusion algorithm, using a modified algorithm* Increase_From_Square *as a subroutine for the entry section, with makespan* $O(\log n \cdot \log(1/\varepsilon))$ *in the model without global clock.*

3.2. Only Number of Processes Known

In this scenario, we build our solution based on the *Probability_Increase* algorithm of [13]. In this algorithm, each process works in $\Theta(\log n)$ phases, each lasting $\Theta(\log(1/\varepsilon))$ rounds. In each round of phase i, a process transmits with probability 2^{-i}.

Lemma 3.2 ([13])**.** *If all processes use the algorithm* Probability_Increase *after being awaken, then there is a successful transmission in time* $k = O(\log n \cdot \log(1/\varepsilon))$ *with probability at least* $1 - \varepsilon$.

We describe how to modify the *Probability_Increase* algorithm to meet the requirements of ε-exclusion. When a process enters the entry section, it first switches to the listening mode and stays in this mode for $k = O(\log n \cdot \log(1/\varepsilon))$ rounds. If within this time it hears another process, it resigns. Afterwards, the process starts to execute the *Probability_Increase* algorithm. Whenever it is not transmitting, it listens, and when it hears a message from another process, it resigns. After executing k rounds of the listening mode and the following k rounds of *Probability_Increase* without resigning, the process enters the critical section. Using this algorithm, the following result can be proved.

Theorem 3.3. *There is an ε-mutual-exclusion algorithm, using a modified algorithm* Probability_Increase *as a subroutine for the entry section, with makespan* $O(\log n \cdot \log(1/\varepsilon))$ *in the KN model.*

Proof. Let k be as defined above in the algorithm definition. Let t be a round in the execution in which there is at least one process in the entry section, no process in the exit or critical section, and such that there was no process in the entry section in the previous round $t - 1$. Let P be the set of processes which are in their entry sections at round $t + k$.

First, we observe that processes which enter their entry section in round $t + k + 1$ or later, i.e., all processes that are not in set P, do not transmit in the time period $[t, t + 2k]$. By Lemma 3.2, with probability $1 - \varepsilon$, there is a process in P which successfully transmits at some round in $[t + k, t + 2k)$. Let $t + k \leq r < t + 2k$ be the first such round, and $p \in P$ be the process transmitting successfully in round r. Note that all other processes being in the entry section resign at this round, and all processes that start their entry sections after round r do not transmit by round $r + k$. Therefore, p does not hear anything before it finishes its *Probability_Increase* subroutine (in the next at most $k - 1$ rounds after r), which implies that it enters the critical section by round $r + k - 1 < t + 2k$. ■

3.3. Only Collision Detection Available

In this scenario, the main idea behind our algorithm is as follows. First, we show how to solve a *static case* of the ε-mutual-exclusion problem, i.e., the case where there is a subset S of processes which start their entry sections at round 1 and no process is activated later. Later, we show that we are then able to solve ε-mutual-exclusion problem in (asymptotically) the same time. In what follows, we assume that whenever a process does not transmit, it listens.

To solve the static case, we first run a simple *Check_If_Single* subroutine, which, with probability at least $1 - \varepsilon$, determines whether there is one active processes or more. In the former case, this process may simply enter the critical section. In the latter, we simulate Willard's algorithm [19], which works in expected time $\log \log n + o(\log \log n)$. The simulation is required, as the original algorithm of [19] assumes that each process can simultaneously transmit and listen in each round. The idea of this simulation is that for each message sent, all listening processes acknowledge it in the next round.

Lemma 3.4. *If there are at least two active processes, it is possible to simulate one round taken in the model in which a process may simultaneously transmit and listen, in two rounds of our model in the setting with collision detection.*

As mentioned above, another building block is a procedure *Check_If_Single*. The algorithm assumes that there is a set of processes which start this procedure simultaneously. The procedure consists of $2 \cdot \log(1/\varepsilon)$ rounds. In each odd round, process i tosses a symmetric coin, i.e., with probability $1/2$ of success, to choose whether it transmits in the current round and listens in the next round, or vice versa. If the process never hears anything, it enters the critical section at the end of the procedure.

Lemma 3.5. *Assume k processes execute the procedure* Check_If_Single. *If $k = 1$, then the only process enters the critical section. If $k \geq 2$, then with probability $1 - \varepsilon$, no process enters the critical section.*

Proof. The first claim holds trivially. For showing the second one, we fix an odd-even pair of rounds. Let E denote the event that there is a process, which does not hear anything in this pair of rounds. For this to happen all processes running *Check_If_Single* have to transmit in the odd round or all have to transmit in the even round. Thus, $\Pr[E] = 2 \cdot 1/2^k = 1/2^{k-1} \leq 1/2$. Since the transmissions in different pairs of rounds are independent, the probability that there exists a process which does not hear anything during the whole algorithm, and thus enters the critical section, is at most $(1/2)^{\log(1/\varepsilon)} = \varepsilon$. ■

We may now describe an algorithm solving the static ε-mutual-exclusion problem. Let S be a subset of processes which simultaneously start their entry sections. In the first $2\log(1/\varepsilon)$ rounds, the processes execute the procedure *Check_If_Single*. Then the processes that did not enter the critical section, run a simulation of Willard's algorithm, as described in Lemma 3.4. The processes that transmit successfully, enter the critical section. Using this algorithm, the following result can be proved.

Theorem 3.6. *In the scenario with collision detection, there is an algorithm solving the static ε-mutual-exclusion problem with expected makespan $O(\log\log n + \log(1/\varepsilon))$.*

Proof. Consider the algorithm described above, based on the procedure *Check_If_Single*. If there is only one process starting its entry section, it enters the critical section right after the procedure *Check_If_Single* (which takes $O(\log(1/\varepsilon))$ rounds). If there is more than one process, with probability $1-\varepsilon$ they do not enter the critical section after this procedure and they all simultaneously start the simulation of Willard's algorithm. By the property of Willard's algorithm [19] and by Lemma 3.4, in expectation there is a successful transmission in $O(\log\log n)$ rounds. ■

It remains to show that we may use an algorithm for static version of ε-mutual-exclusion to solve the general version of the ε-mutual-exclusion problem. The idea is to synchronize processes at the beginning, and then to transmit a "busy" signal in every second round. New processes starting their entry section note this signal and will not compete for the critical section, until an exit section releases the shared channel.

Theorem 3.7. *If there exists an algorithm* Alg *for the static ε-mutual-exclusion problem with (expected) makespan T in the model with collision detection, then there exists an algorithm* Alg′ *for the ε-mutual-exclusion problem with (expected) makespan $2 + 2\cdot T$ in the same setting.*

4. Fairness

The algorithms shown in [8] and Section 3 do not consider the no-lockout property, i.e., it may happen that a process never gets out of its entry section, as other processes exchange access to the critical section among themselves. We show how to modify algorithms satisfying no-deadlock property (in particular, the algorithms from [8]), so that the no-lockout property is fulfilled. Moreover, our transformation allows to express the (expected) makespan of obtained fair protocols in terms of the (expected) makespan of the original weaker protocols.

Each process maintains an additional local counter of losses. When it starts its entry section, it sets its counter to zero and whenever it loses the competition for the critical section, i.e., when some other process enters the critical section, it increases this counter by one. When a process enters its exit section, it becomes a guard: it helps processes currently being in the entry section to choose one of them with the highest loss counter. How high the loss counter can grow is bounded by the number of processes in their entry sections at the moment when the considered process entered its current entry section. Thus, also the time after which the process will enter the critical section is bounded.

Lemma 4.1. *If either collision detection is available or the number of nodes is known, it is possible to transform a mutual exclusion algorithm with (expected) makespan T into* *an algorithm, which also guarantees the no-lockout property and has (expected) makespan* $O(T + \log n)$.

Proof. Here we only describe a transformation for the CD scenario; the analysis and the variant for KN will appear in the full version of the paper. Let Alg be a given subroutine for the entry section, satisfying no-deadlock property. In order to use it for an entry procedure satisfying stronger no-lockout property, we slow down algorithm Alg three times, by preceding each original round by two additional rounds: in the first one the process transmits signal 1, while in the second one it only listens. We call the obtained subroutine Alg$'$.

We also need the following selection subroutine. Assume there is a single guard and a subset (may be empty) of other processes, called *competing processes.* They all start the selection subroutine in the same round. The goal is to elect one of the competing processes to enter the critical section. The subroutine is partitioned into *blocks*, each consisting of three rounds. In the first two rounds of each block only the guard transmits, and the signals are 1 and 0, respectively. The purpose of these rounds is to assure that processes that start their entry sections later will not disturb the selection subroutine. The competition, which is essentially a binary search for the highest loss counter of the competing processes, proceeds in the following phases. In the ith block of the first phase all processes whose loss counter is at least 2^i broadcast a 0 (after the guard's 10), all other processes listen. The phase ends with a block i when silence is heard, thus all competitors and the guard know that the highest loss counter is between 2^{i-1} and 2^i. Then a binary search is performed in additional $O(i)$ blocks in similar manner. Additional binary search is performed to choose one process (the one with the minimum id) from all processes with the same maximal number of losses.

We now describe a procedure governing the exit section. Recall that a process being in the critical section always broadcasts the critical message to let others know that the channel is occupied. For the purpose of this reduction and its analysis, we denote the critical message by a single bit 1 (this is only technical assumption to simplify the proof arguments). When the process starts its exit section and becomes a guard, it transmits a 0 in the first round and listens in the second round. If the guard hears silence then it switches to the remainder section; otherwise it participates in the selection subroutine described above.

Each process starting its entry section listens for three rounds. If it hears silence during all these rounds, it starts executing Alg$'$ until some process enters the critical section (it is guaranteed by no-deadlock property of Alg, and can be extended to Alg$'$ as well); then it resets its state and starts again its entry section procedure with round counter 1. It also resets its state and starts again with round counter 1 in case it hears anything different from $1, 1, 1$ and $1, 1, 0$ during the first three rounds of listening. In the remaining third case, i.e., when the process has heard $1, 1, 1$ or $1, 1, 0$, it keeps listening until the first round t, counting from the first listening round in this run, such that the process has heard signals $1, 1, 0$ in rounds $t - 2, t - 1, t$, respectively. It then transmits in round $t + 1$ and starts the selection subroutine in round $t + 2$. ■

By combining Lemma 4.1 with the results from Section 3 and with the existing no-deadlock deterministic algorithms of [8], we obtain the following two conclusions.

Corollary 4.2. *There exists a randomized algorithm with expected makespan* $O(\log n + \log(1/\varepsilon))$ *solving the* ε*-mutual-exclusion problem in the model in which collision detection is available, and a randomized algorithm with makespan* $O(\log n \cdot \log(1/\varepsilon))$ *in the KN model.*

Corollary 4.3. *There exists a deterministic algorithm with makespan $O(\log n)$ solving the mutual exclusion problem in the model in which collision detection is available and a deterministic algorithm with makespan $O(n \log^2 n)$ in the KN model.*

References

[1] H. Attiya, J. Welch, *"Distributed Computing,"*, 2004, John Wiley and Sons, Inc.

[2] R. Bar-Yehuda, O. Goldreich, A. Itai, On the time complexity of broadcast in radio networks: an exponential gap between determinism and randomization, *Journal of Computer and System Sciences*, **45** (1992) 104–126.

[3] M.A. Bender, M. Farach-Colton, S. He, B.C. Kuszmaul, C.E. Leiserson, Adversarial contention resolution for simple channels, in *Proceedings, 17th Annual ACM Symposium on Parallel Algorithms (SPAA)*, 2005, pp. 325–332.

[4] J. Capetanakis, Tree algorithms for packet broadcast channels. *IEEE Transactions on Information Theory* **25** (1979) 505–515.

[5] B.S. Chlebus, L. Gasieniec, D.R. Kowalski, T. Radzik, On the wake-up problem in radio networks, in *Proceedings, 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, 2005, pp. 347–359.

[6] B.S. Chlebus, D.R. Kowalski, M.A. Rokicki, Adversarial queuing on the multiple-access channel, in *Proceedings, 25th ACM Symposium on Principles of Distributed Computing (PODC)*, 2006, pp. 92–101.

[7] A.E.F. Clementi, A. Monti, R. Silvestri, Selective families, superimposed codes, and broadcasting on unknown radio networks, in *Proceedings, 12th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2001, pp. 709—718.

[8] J. Czyzowicz, L. Gasieniec, D.R. Kowalski, A. Pelc, Consensus and mutual exclusion in a multiple access channel, in *Proceedings, 23rd International Symposium on Distributed Computing (DISC)*, 2009, pp. 512–526.

[9] L. Gasieniec, A. Pelc, D. Peleg, The wakeup problem in synchronous broadcast systems, *SIAM Journal on Discrete Mathematics*, **14** (2001) 207–222.

[10] L.A. Goldberg, M. Jerrum, S. Kannan, M. Paterson, A bound on the capacity of backoff and acknowledgment-based protocols, *SIAM Journal on Computing* **33** (2004) 313–331.

[11] A.G. Greenberg, S. Winograd, A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. *J. ACM* **32** (1985) 589–596.

[12] T. Jurdzinski, M. Kutylowski, J. Zatopianski, Efficient algorithms for leader election in radio networks, in *Proceedings, 21st ACM Symposium on Principles of Distributed Computing (PODC)*, 2002, pp. 51–57.

[13] T. Jurdziński, G. Stachowiak, Probabilistic algorithms for the wakeup problem in single-hop radio networks, in *Proceedings, 13th International Symposium on Algorithms and Computation (ISAAC),* 2002, LNCS 2518, pp. 535–549.

[14] D.R. Kowalski, On selection problem in radio networks, in *Proceedings, 24th ACM Symposium on Principles of Distributed Computing (PODC)*, 2005, pp. 158–166.

[15] Y. Kushilevitz, Y. Mansour, An $\Omega(D \log(N/D))$ lower bound for broadcast in radio networks, *SIAM Journal on Computing* **27** (1998) 702–712.

[16] N.A. Lynch, Distributed Algorithms, Morgan Kaufmann Publ., Inc., 1996.

[17] K. Nakano, S. Olariu, Uniform leader election protocols for radio networks, *IEEE Transactions on Parallel Distributed Systems* **13** (2002) 516–526.

[18] B.S. Tsybakov, V.A. Mikhailov, Free synchronous packet access in a broadcast channel with feedback, *Prob. Inf. Transmission* **14** (1978) 259–280. (Translated from Russian original in *Prob. Peredach. Inf.*, 1977.)

[19] D.E. Willard, Log-logarithmic selection resolution protocols in a multiple access channel, *SIAM Journal on Computing* **15** (1986) 468–477.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 95-106
www.stacs-conf.org

EXACT COVERS VIA DETERMINANTS

ANDREAS BJÖRKLUND

E-mail address: andreas.bjorklund@yahoo.se

Abstract. Given a k-uniform hypergraph on n vertices, partitioned in k equal parts such that every hyperedge includes one vertex from each part, the k-Dimensional Matching problem asks whether there is a disjoint collection of the hyperedges which covers all vertices. We show it can be solved by a randomized polynomial space algorithm in $O^*(2^{n(k-2)/k})$ time. The $O^*()$ notation hides factors polynomial in n and k.

The general Exact Cover by k-Sets problem asks the same when the partition constraint is dropped and arbitrary hyperedges of cardinality k are permitted. We show it can be solved by a randomized polynomial space algorithm in $O^*(c_k^n)$ time, where $c_3 = 1.496, c_4 = 1.642, c_5 = 1.721$, and provide a general bound for larger k.

Both results substantially improve on the previous best algorithms for these problems, especially for small k. They follow from the new observation that Lovász' perfect matching detection via determinants (Lovász, 1979) admits an embedding in the recently proposed inclusion–exclusion counting scheme for set covers, *despite* its inability to count the perfect matchings.

1. Introduction

The Exact Cover by k-Sets problem (XkC) and its constrained variant k-Dimensional Matching (kDM) are two well-known NP-hard problems. They ask, given a k-uniform hypergraph, if there is a subset of the hyperedges which cover the vertices without overlapping each other. In the kDM problem the vertices are further partitioned in k equal parts and the hyperedges each includes exactly one vertex from each part. While being two of the 21 items of Karp's classic list of NP-complete problems [6] for $k \geq 3$, little is known on their algorithmic side. In this paper, we present stronger worst case time bounds for these problems by combining Lovász' perfect matching detection algorithm via determinants [10] with the inclusion–exclusion counting for set covers [1]. We show

Theorem 1.1. *k-Dimensional Matching on n vertices can be solved by a Monte Carlo algorithm with exponentially low probability of failure in n, using space polynomial in n, running in $O^*(2^{n(k-2)/k})$ time.*

Theorem 1.2. *Exact Cover by k-Sets on n vertices can be solved by a Monte Carlo algorithm with exponentially low probability of failure in n, using space polynomial in n,*

1998 ACM Subject Classification: F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Hypergraphs.

Key words and phrases: Moderately Exponential Time Algorithms, Exact Set Cover, k-Dimensional Matching.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2447

Algorithm \ k	3	4	5	6	7	8
kDM in [1]	1.587	1.682	1.741	1.782	1.811	1.834
kDM here	1.260	1.414	1.516	1.587	1.641	1.682
XkC in [1]	1.842	1.888	1.913	1.929	1.940	1.948
XkC in [8]	1.769	1.827	1.862	1.885	1.901	1.914
XkC here	1.496	1.642	1.721	1.771	1.806	1.832

Table 1: Comparison of the base c in the $O^*(c^n)$ runtime of previous and the new algorithms.

running in $O^*(c_k^n)$ *time, with* $c_3 = 1.496, c_4 = 1.642, c_5 = 1.721, c_6 = 1.771, c_7 = 1.806$*, and in general* $c_k < 2\left(8.415k^{0.9-k}(k-1)^{0.6}(k-1.5)^{k-1.5}\right)^{-1/k}$

These bounds are large improvements over the previously known ones. In particular, for three dimensional matching our algorithm runs in time asymptotically proportional to the square root of the previous best algorithm's runtime.

We hope the present paper conveys the message that inclusion–exclusion is amendable not only to counting problems, but can at times be used more directly to settle the decision version of a problem.

1.1. Previous Work

Perhaps the most famous algorithmic contribution on the subject of exact covers is Knuth's *Dancing Links* paper [7], which actually just addresses a general implementation issue which saves a small constant factor in the natural backtracking algorithm for the problem. About the backtracking approach on exact cover he writes "Indeed, I can't think of any other reasonable way to do the job in general". While we certainly may agree depending on how much you put in the words "reasonable" and "general", we must point out that the best provable worst case bounds for the problems are obtained by analyzing very different algorithms. Björklund et al. [2] uses inclusion–exclusion and fast zeta transforms on the full subset lattice to show that exact set covers of any n vertex hypergraph can be counted in $O^*(2^n)$ time even when the number of hyperedges to choose from are exponential. Restricted to k-uniform hypergraphs, Koivisto [8] proposes a simple clever dynamic programming over subsets which show that Exact Cover by k-Sets can be solved in $O^*(2^{n(2k-2)/\sqrt{(2k-1)^2-2\ln(2)}})$ time. The algorithm is actually capable of counting the solutions and also works for not necessarily disjoint covers. It does, however, use exponential space. The best previous algorithm for the problem using only polynomial space is given in [1] and has a runtime bound in $O^*((1+k/(k-1))^{n(k-1)/k})$. For k-Dimensional Matching, the best known algorithm as far as we know is an $O^*(2^{n(k-1)/k})$ time algorithm resulting from a generalization of Ryser's inclusion–exclusion counting formula for the permanent [12], presented in [1]. A comparison of the bounds guaranteed by these algorithms and the ones given in this paper is shown in Table 1 for small k.

For $k = 2$ the problems X2C and 2DM are better known as the problems of finding a perfect matching in a general and bipartite graph, respectively. For these problems several polynomial time algorithms are known. We definitely admit that it seems like an obvious idea to try to reduce the $k > 2$ cases to the $k = 2$ case searching for faster algorithms for larger k. Still, we believe that it is far from clear how to achieve this efficiently. In this paper we make such an attempt by reducing the $k > 2$ cases to variants of one of the first polynomial time algorithms for detecting the existence of perfect matchings: Lovász' algorithm from [10] which evaluates the determinant of the graph's Tutte matrix [13] at a random point.

2. Our Approach

2.1. Preliminaries

We use the terminology of (multi)hypergraphs. A hypergraph $H = (V, E)$ is a set V of n vertices and a *multiset* E of (hyper)edges which are subsets of V. Note in particular that with this definition edges may include only one (or even no) vertex and may appear more than once. In a k-uniform hypergraph each edge $e \in E$ has size $|e| = k$. Given a vertex subset $U \subseteq V$, the *projected hypergraph* of $H = (V, E)$ on U, denoted $H[U] = (U, E[U])$ is a hypergraph on U where there is one edge e_U in $E[U]$ for every $e \in E$, defined by $e_U = e \cap U$, i.e. the projection of e on U.

We study two related problems.

Definition 2.1 (k-Dimensional Matching, kDM)**.**

Input: A k-uniform hypergraph $H = (V_1 \cup V_2 \cup \cdots V_k, E)$, with $E \subseteq V_1 \times V_2 \times \cdots V_k$.
Question: Is there $S \subseteq E$ s.t. $\cup_{s \in S} s = V_1 \cup V_2 \cup \cdots V_k$ and $\forall s_1 \neq s_2 \in S : s_1 \cap s_2 = \emptyset$.

Definition 2.2 (Exact Cover by k-Sets, XkC)**.**

Input: A k-uniform hypergraph $H = (V, E)$.
Question: Is there $S \subseteq E$ s.t. $\cup_{s \in S} s = V$ and $\forall s_1 \neq s_2 \in S : s_1 \cap s_2 = \emptyset$.

For a matrix $\mathbf{A}$ we will by $\mathbf{A}_{i,j}$ denote the entry at row i and column j.

2.2. Determinants

The determinant of an $n \times n$-matrix $\mathbf{A}$ over an arbitrary ring R can be defined by the Leibniz formula

$$\det(\mathbf{A}) = \sum_{\sigma:[n]\to[n]} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} \mathbf{A}_{i,\sigma(i)} \tag{2.1}$$

where the summation is over all permutations of n elements, and sgn is a function called the sign of the permutation which assigns either one or minus one to a permutation. In this paper we will restrict ourselves to computing determinants over fields of characteristic two, $\mathrm{GF}(2^m)$ for some positive integer m. In such fields every element serves as its own additive inverse, and in particular so does the element one, and the sgn function identically maps one to every permutation. Thus it vanishes from Eq. 2.1 in this case, and the determinant coincides with another matrix quantity, called the *permanent*:

$$\operatorname{per}(\mathbf{A}) = \sum_{\sigma:[n]\to[n]} \prod_{i=1}^{n} \mathbf{A}_{i,\sigma(i)} \tag{2.2}$$

Permanents of 0–1-matrices over the natural numbers are known to count the perfect matchings of the bipartite graph described by the matrix. The reader may subsequently be tempted to think that this identity of determinants and permanents over fields of characteristic two is the property that makes our algorithms work. There is however nothing magical about these fields in this context. Our reason for working in $\mathrm{GF}(2^m)$ is simply that with this choice of fields we don't even have to define the sign function, making several of the proof arguments later on much easier to digest. In principle though, *any* large enough field will work, with slightly more complicated proofs.

The interesting property of the determinant that we will exploit here is that although it is defined above in Eq. 2.1 as a sum of an exponential number of terms, it admits computation in time polynomial in n. This can be achieved for instance via the so called LU-factorization of the matrix which almost any textbook on linear algebra will tell you. In fact, computing the determinant is no harder than square matrix multiplication, see [3], and hence it can be done in $O(n^\omega)$ field operations where $\omega = 2.376$ is the Coppersmith–Winograd exponent [4].

2.3. Inclusion–Exclusion for Set Covers

Let us review the inclusion–exclusion counting scheme for exact set covers presented by Björklund and Husfeldt in [1]: Given a k-uniform hypergraph $H = (V, E)$ and any subset $U \subseteq V$, we can count the number of Exact Covers by k-Sets, denoted $\#\mathrm{X}k\mathrm{C}(H)$, by the inclusion–exclusion formula

$$\#\mathrm{X}k\mathrm{C}(H) = \sum_{X \subseteq V-U} (-1)^{|X|} W(H,U,X) \tag{2.3}$$

where $W(H,U,X)$ counts the number of ways to exactly cover U with $|V|/k$ edges in $H[U]$ whose corresponding edges in H are disjoint from X. Put differently, $W(H,U,X)$ counts the number of ways to pick $|V|/k$ edges from H, all having an empty intersection with X, which cover U without any overlap. In particular, when $U = \emptyset$ it is straightforward to compute $W(H,\emptyset,X)$ by just counting the number of edges in H disjoint from X, calling this quantity $d(X)$, and then computing the binomial $\binom{d(X)}{m}$. In [1], some examples where this algorithm could be accelerated by choosing a larger U were identified where the speedup was obtained by utilizing U's such that the projected hypergraph on U had low path–width. This enabled efficient counting by dynamic programming over a path decomposition.

2.4. Moving to GF(2^m)

In this paper, we find a new way to allow a large U to expedite the computation of the formula Eq. 2.3 above. We observe that whenever the projected hypergraph contains edges of size at most two, we can use determinants to compute the formula faster. We note that if the problem of counting perfect matching had an efficient algorithm A, we would almost immediately get an $O^*(2^{n(k-2)/k})$ time algorithm for the kDM problem. We would simply let U be any two of the parts in the input partition, and use A to compute $W(H,U,X)$. Unfortunately, counting perfect matchings even in bipartite graphs is #P-complete [14].

The key insight of the present paper circumvents the apparent obstacle formed by the intractability of counting matchings: we only need to be able to efficiently compute *some* fixed weighted sum of the matchings (with no weights set to zero). This is exactly where the determinants come to our rescue. The price we pay is that we have to give up counting the solutions over the natural numbers. Here we demonstrate the result through counting over fields of characteristic two which only allow us to detect if there is a cover at all and gives us little knowledge of their number. Furthermore, to avoid having an even number of solutions cancel we will employ a fingerprint technique, very much in the same spirit as Williams [15] recently extended the k-path detection algorithm based on an algebraic sieving method of Koutis [9]. The fingerprint idea is to think of the computation as evaluating a polynomial of a degree much smaller than the number of elements of its base field and then computing it at a randomly chosen point. The fact that a polynomial cannot have more roots than its degree assure us that with great probability we discover with this single point probing whether the polynomial is the zero-polynomial or not. We will in fact use the multivariate polynomial analogue, see e.g. [11].

Lemma 2.3 (Schwartz-Zippel)**.** *Let $P(x_1, x_2, ..., x_n)$ be a non-zero n-variate polynomial of degree d over a field F. Pick $r_1, r_2, ..., r_n \in F$ uniformly at random, then*

$$Pr(P(r_1, r_2, ..., r_n) = 0) \leq \frac{d}{|F|}$$

For now, it is sufficient to think of the inclusion–exclusion formula of Eq. 2.3 as evaluating a multivariate polynomial over the base field GF(2^m) for some m. In what follows we will associate with all edges e in the input hypergraph a variable v_e. Our modified version of Eq. 2.3 reads as follows.

Lemma 2.4. *Given an XkC-instance $H = (V, E)$ and the family of all its solutions $\mathcal{S}$, we have that, for every subset $U \subseteq V$,*

$$\sum_{X \subseteq V-U} W_{2,f}(H, U, X) = \sum_{E' \in \mathcal{S}} \prod_{e \in E'} v_e^{f(e)} \tag{2.4}$$

where the computation is over a multivariate polynomial ring over $GF(2^m)$, f is a function mapping the edges to the positive integers, and

$$W_{2,f}(H, U, X) = \sum_{E''} \prod_{e \in E''} v_e^{f(e)} \tag{2.5}$$

where the summation is over all $E'' \subseteq E$, satisfying four constraints

- *Avoidance, $\forall e \in E'' : e \cap X = \emptyset$*
- *Cardinality, $|E''| = |V|/k$*
- *Coverage, $U \subseteq \cup_{e \in E''} e$*
- *Disjointness, $\forall e_1 \neq e_2 \in E'' : e_1 \cap e_2 \cap U = \emptyset$*

Proof. First, note that every $E' \in \mathcal{S}$ fulfills all four conditions Avoidance, Cardinality, Coverage, and Disjointness for $X = \emptyset$, but violates Avoidance for every other X, irrespective of the choice of U. Thus, the contribution $\prod_{e \in E'} v_e$ of every solution E' is counted precisely once.

Second, a non-solution E'' obeying the three conditions Cardinality, Coverage, and Disjointness, fulfills the Avoidance condition for an *even* number of choices of X irrespective of U, namely for all subsets of the elements of V that the union of the sets in E'' fails to cover. Hence, all of these contributions $\prod_{e \in E''} v_e^{f(e)}$ cancel each other since we are working in a field of characteristic two. ■

Combining the two Lemmas above 2.3 and 2.4 into an algorithm choosing a random point $r_1, r_2, ..., r_{|E|} \in$ GF(2^m) and evaluating the left-hand sum of Eq. 2.4 in the straightforward fashion, we get:

Corollary 2.5. *Given an XkC-instance $H = (V, E)$ and a subset $U \subseteq V$, there is a Monte Carlo algorithm which returns "No" whenever there is no cover and returns "Yes" with* *probability at least* $1 - \max_{e \in E} f(e)|V|/(k2^m)$ *when there exists at least one, running in time* $O^*(2^{|V|-|U|}\tau(W_{2,f}, U))$*, where* $\tau(W_{2,f}, U)$ *is the time required to evaluate any of the polynomials* $W_{2,f}(H, U, X)$ *for* $X \subseteq V - U$*, in a random point over the base field* $GF(2^m)$.

Note that by letting m be in the order of n, when f is bounded by a constant, we get exponentially low probability of failure in n. Armed with Corollary 2.5, we can start looking for projections U over which the computation of $W_{2,f}(H, U, X)$ is easy. The next two sections will describe two examples of how we can use determinants to accelerate the computation.

3. k-Dimensional Matching

We begin by the easier application, kDM. For this problem we can trivially find a large vertex subset on which the projected instance is a multigraph, and in fact also bipartite: we just use any two of the parts in the vertex partition given as input. Edmonds [5] observed that one could relate a bipartite graphs' perfect matchings to the determinant of a symbolic matrix. A perfect matching is a collection of disjoint edges so that every vertex is covered by precisely one edge. To a given a bipartite graph $G = (U, V, E), n = |U| = |V|$, he associated an $n \times n$-matrix $\mathbf{A}$ with rows representing vertices in U, and columns the vertices of V, and equated $\mathbf{A}_{i,j}$ with a variable v_{ij} if $(i, j) \in E$ and zero otherwise. He showed that the determinant of $\mathbf{A}$ is non-zero iff G has a perfect matching. We will use essentially the same result, with the small exception that we need to deal with multiple edges between a vertex pair, making sure all contributes. Formally

Definition 3.1. Given a hypergraph $H = (V, E)$ and a subset $U \subseteq V$ such that the projected hypergraph $H[U]$ is a bipartite multigraph on two equally sized vertex parts $U_1 \cup U_2 = U$, its Edmonds matrix, denoted $\mathbf{E}(H, U_1, U_2)$, is defined by

$$\mathbf{E}(H, U_1, U_2)_{i,j} = \sum_{\substack{e=(i,j) \in E[U] \\ i \in U_1, j \in U_2}} v_e$$

where again, v_e is a variable associated with the edge e.

We formulate our Lemma in terms of a special case of XkC instead of kDM directly to capture a more general case.

Lemma 3.2. *For a X*k*C-instance* $H = (V, E)$ *and two equally sized disjoint vertex subsets* $U_1, U_2 \subseteq V$ *such that the projected hypergraph* $H[U_1 \cup U_2]$ *is a bipartite multigraph,*

$$\det(\mathbf{E}(H, U_1, U_2)) = \sum_{M \in \mathcal{M}} \prod_{e \in M} v_e \tag{3.1}$$

where the computation is over a multivariate polynomial ring over GF(2^m*) for some* m *and the summation is over all perfect matchings* $\mathcal{M}$ *in* $H[U_1 \cup U_2]$.

Proof. By definition of the determinant 2.1, the summation is over all products of n of the matrix elements in which every row and column are used exactly once. Transferred to the associated bipartite graph, this corresponds to a perfect matching in the graph since rows and columns represent the two vertex sets respectively. Moreover, the converse is also true, i.e. for every perfect matching there is a permutation describing it. Hence the mapping is one-to-one. The inner product counts all choices of edges producing a matching described by a permutation σ since:

$$\prod_{i=1}^{n} \mathbf{E}_{i,\sigma(i)} = \prod_{i=1}^{n} \sum_{e=(i,\sigma(i))} v_e = \sum_{M\in\mathcal{M}(\sigma)} \prod_{e\in M} v_e \tag{3.2}$$

where $\mathcal{M}(\sigma)$ is the set of all perfect matchings $e_1, e_2, ..., e_n$ such that $e_i = (i, \sigma(i))$. ■

3.1. The Algorithm

Now we are ready to prove Theorem 1.1. Given an input instance $H = (V_1, V_2, ..., V_k, E)$ to the kDM problem where $V_1, V_2, ..., V_k$ describe the vertex partition of the n vertices, we simply let $U = V_1 \cup V_2$ in the algorithm described by Corollary 2.5, with f mapping one to every edge. To compute $W_{2,f}(H, U, X)$ we construct the Edmonds matrix of the hypergraph H *restricted* to its edges disjoint to X, projected on U, with the variables replaced by the random sample point $(r_1, r_2, ..., r_{|E|})$ chosen. Next we compute its determinant. The correctness follows from Lemma 3.2, after noting that every perfect matching in a projected hypergraph contains n/k disjoint edges. The runtime bound is easily seen to be $O^*(2^{n(k-2)/k})$ since $|U| = |V_1| + |V_2| = 2n/k$.

4. Exact Cover by k-Sets

Next we proceed to the XkC problem. In comparison to the kDM we are faced with a number of additional obstacles on our way to a similar result.

- First, a projection will typically capture edges differently, some will have large projections and some no at all.
- Second, in particular the projected edges will probably not form a multigraph.
- Third, even if they did it may not be a bipartite one.

For the first obstacle, we will prove that it is sufficient to find a projection on which at least one cover's edges all leave projected edges of size two *or less*. This is basically an extension of the idea for the XkC algorithm in proposition 10 in [1]. There, a vertex subset U is picked uniformly at random of a carefully chosen size, and in the projected hypergraph only the edges which leave a projection of size one or less are kept. Then the inclusion–exclusion formula Eq. 2.3 is used after noting that $W(H, U, X)$ is now easy to compute. The process is repeated a number of times dictated by the size of U. The best size to use is a trade-off of the resulting summation runtime and the probability that a cover is projected gracefully in the sense that all its edges are kept after the projection.

For the second obstacle, in addition to handling multiple edges we also need to count perfect matchings in which loops, i.e. edges connecting a vertex to itself, count as covering the vertex of its endpoints. Since this means that not all perfect matchings will involve the same number of edges, we have to take special care to make the determinants useful. We use polynomial interpolation to solve for the contributions of matchings of the same size separately to be able to fulfill the Cardinality constraint for $W_{2,f}$ in Corollary 2.5. To this end we introduce an auxiliary variable s parametrizing the matrices and use several determinant calculations.

For the third obstacle, we will use a variation of a result generalizing Edmonds' due to Tutte [13]. He showed that even for general not necessarily bipartite graphs one can make a connection between its perfect matchings and the determinant of a symbolic matrix, although twice as large matrices in both directions are required. To a given a graph $G = (V, E), n = |V|$ he associates an $n \times n$-matrix $\mathbf{A}$ with rows and columns representing the vertices, and assigns $\mathbf{A}_{i,j} = v_{i,j}$ for $i < j$ and $\mathbf{A}_{i,j} = -v_{i,j}$ for $i > j$ with $v_{i,j}$ a variable for each edge $(i, j) \in E$. The remaining entries are set to zero. The determinant of $\mathbf{A}$ is non-zero iff G has a perfect matching.

We define matrices similar to Tutte's:

Definition 4.1. Given a hypergraph $H = (V, E)$ and a subset $U \subseteq V$ such that in the projected hypergraph $H[U]$ all edges have size at most two, its Tutte matrix of index s, denoted $\mathbf{T}^{(s)}(H, U)$, is defined by

$$\mathbf{T}^{(s)}(H,U)_{i,j} = \begin{cases} \sum v_e & : e \in E[U], e = (i,j), i \neq j \\ s \sum v_e & : e \in E[U], e = (i,j), i = j \end{cases}$$

Lemma 4.2. *For a XkC-instance $H = (V, E)$ and a vertex subset $U \subseteq V$ such that in the projected hypergraph $H[U]$, every edge has size at most two,*

$$\det(\mathbf{T}^{(s)}(H,U)) = \sum_{M \in \mathcal{M}} s^{\Lambda(M)} \prod_{e \in M} v_e^{p(e)} \tag{4.1}$$

where the computation is over a multivariate polynomial ring over GF(2^m) for some m, the summation is over all perfect matchings $\mathcal{M}$ in $H[U]$, $\Lambda(M)$ is the number of loops in the matching M, and $p(e) = 1$ if e is a loop and $p(e) = 2$ otherwise.

Proof. By definition of the determinant 2.1, the summation is over all products $\prod_{i=1}^{n} \mathbf{T}^{(s)}_{i,\sigma(i)}$ for a permutation σ. Call a permutation σ good if $\forall i : \sigma(\sigma(i)) = i$ holds, and bad otherwise. We will argue that only good permutations contribute to the sum. To see why, consider a bad σ. Then there exists a smallest i such that $\sigma(\sigma(i)) = j \neq i$. Look at the cyclic sequence $\{c_i\}$ where $c_0 = i$ and $c_{k+1} = \sigma(c_k)$ for $k > 0$. Let $L > 2$ be the smallest positive integer such that $c_L = i$ (Note that there must be one and that all c_i in between must by distinct since every element in 1 through n is mapped to exactly once). Next define a cycle reversal operation D mapping bad permutations on bad permutations by letting $D(\sigma)$ be identical to σ except in the points c_1 through c_L, where instead $D(\sigma)(c_i) = c_{i-1}$. Now first observe that the reversal operation is dual in the sense that $D(D(\sigma)) = \sigma$ and that $D(\sigma) \neq \sigma$ since $L > 2$, and hence every bad permutation can be uniquely paired with another bad permutation. Second note that the contribution of a bad permutation is identical to the contribution of its dual, since the Tutte matrices are symmetrical. Thus, since we are counting in a field of characteristic two, they cancel each other.

Next we continue to observe that the good permutations describe precisely the structure of all possible perfect matchings in a multigraph: i's such that $\sigma(i) \neq i$ describe ordinary two-vertex edges in the matching, and i's such that $\sigma(i) = i$ describe loops.

The inner product of Eq. 2.1 reads

$$\prod_{i=1}^{n} \mathbf{T}^{(s)}_{i,\sigma(i)} = \left(\prod_{i, i=\sigma(i)} s \sum_{e=(i,i)} v_e \right) \left(\prod_{i, i \neq \sigma(i)} \sum_{e=(i,\sigma(i))} v_e \right) = \sum_{M \in \mathcal{M}(\sigma)} s^{\Lambda(M)} \prod_{e \in M} v_e \tag{4.2}$$

where $\mathcal{M}(\sigma)$ is the set of all *directed* perfect matchings $e_1, e_2, ..., e_n$ described by the good permutation σ for which $e_i = (i, \sigma(i))$.

Now consider a directed perfect matching $e_1, e_2, ..., e_n$ such that for some j, $e_j \neq e_{\sigma(j)}$, and refer to it as being bad. We will see that all of these cancel in very much the same way as the bad permutations did. Namely, again find the smallest j for which this is the case, and define a reversal operation R_σ mapping bad directed perfect matchings onto themselves by exchanging e_j and $e_{\sigma(j)}$. Since this operation pairs up the bad directed perfect matchings ($R_\sigma(\{e_i\}) \neq \{e_i\}$ and $R_\sigma(R_\sigma(\{e_i\})) = \{e_i\}$) and we work in a field of characteristic two, their contributions cancel. Thus we are left with only good permutations and good directed perfect matchings. The latter can be thought of as undirected perfect matchings in which every non-loop edge is included twice in the product. ■

To find the contributions of matchings of the same size separately, think of the matchings partitioned in groups $\mathcal{M}_0, \mathcal{M}_1, ..., \mathcal{M}_n$ according to the number of loops of the matching. We can rewrite the determinant in Lemma 4.2 as

$$\det(\mathbf{T}^{(s)}(H,U)) = \sum_{i=0}^{n} s^i M_i \tag{4.3}$$

where $M_i = \sum_{M \in \mathcal{M}_i} \prod_{e \in M} v_e^{p(e)}$ are the quantities we seek. The right hand side of Eq. 4.3 is a degree n polynomial in s and thus we can solve for $M_0, M_1, ..., M_n$ by computing $\det(\mathbf{T}^{(s)}(H,U))$ in n different choices of s, and use Lagrange's interpolation formula to recover the sought values. In fact, either there are no matchings with an odd number of loops or no matchings with an even number of loops depending on the parity of $|U|$. Consequently, the evaluation of $n/2$ points suffices, but we disregard from this optimization possibility for simplicity. Once we have the M_i's we are close to be able to compute $W_{2,f}$ efficiently according to the following Lemma:

Lemma 4.3. *Given a XkC instance $H = (V,E)$ and a $U \subseteq V$ such that for all edges $e \in E$,$|e \cap U| \leq 2$, $f(e) = 2$ if $|e \cap U| = 2$ and 1 otherwise, and any $X \subseteq V - U$,*

$$W_{2,f}(H,U,X) = \sum_{i=0}^{|U|} Z(\frac{|V|}{k} - \lfloor\frac{|U|+i}{2}\rfloor)M_i \tag{4.4}$$

where $M_i = \sum_{M \in \mathcal{M}_i} \prod_{e \in M} v_e^{p(e)}$ are the contribution of all matchings $\mathcal{M}_i$ containing exactly i loops in the projected hypergraph of H on U restricted to the edges disjoint to X, and

$$Z(i) = \sum_{\substack{E' \subseteq Z \\ |E'|=i}} \prod_{e' \in E'} v_{e'} \tag{4.5}$$

with Z defining the set of edges e disjoint to X also having an empty intersection with U.

Proof. The M_i's count the contribution of all ways to cover U with the edges which leaves a non-empty projection on U and the $Z(i)$'s count the contribution of all ways to choose edges leaving an empty projection. Note that a matching from $\mathcal{M}_i$ involves exactly $\lfloor\frac{|U|+i}{2}\rfloor$ edges if it exists. The right hand side of Eq. 4.4 convolutes over all ways their total number of edges could equal $|V|/k$ in order to meet the Cardinality constraint in Corollary 2.5. ■

The only piece missing is a simple way to evaluate $Z(i)$, and we note that it can be done by dynamic programming through a simple recursion. Number the edges in Z defined in Lemma 4.3 arbitrarily as $e_1, e_2, ..., e_p$, set $Z_i = \{e_1, e_2, ..., e_i\}$, and define

$$z(i,j) = \sum_{\substack{E' \subseteq Z_j \\ |E'|=i}} \prod_{e' \in E'} v_{e'} \tag{4.6}$$

These can be solved for by

$$z(i,j) = \begin{cases} 1 & : i = j = 0 \\ 0 & : i = 0 \text{ or } j = 0 \\ z(i-1, j-1)v_{e_j} + z(i, j-1) & : \text{otherwise} \end{cases} \tag{4.7}$$

and we finally compute $Z(i)$ through $Z(i) = z(i,p)$.

4.1. The Algorithm

We are ready to prove Theorem 1.2. First we describe the algorithm. Given an input instance $H = (V,E)$ to the XkC problem, we compute two parameters t and I depending on k. These are given by the calculations in the next section 4.2. We repeat the following procedure until we detect a cover, in which case we report so, or have tried unsuccessfully I times, in which case we report that no cover was found:

Algorithm 4.4.

(1) Choose a tn-sized subset $U \subseteq V$ uniformly at random.
(2) Construct $H_U = (V, E_U)$ where $E_U = \{e | e \in E, |e \cap U| \leq 2\}$.
(3) Run the summation algorithm in Corollary 2.5 on H_U, using U, and let $f(e) = 2$ if $|e \cap U| = 2$ and 1 otherwise. Use the method of the previous section 4 to compute $W_{2,f}(H_U, U, X)$, i.e.
  (a) Construct the Tutte matrices $\mathbf{T}^{(s)}$ of $H_U[U]$ restricted to its edges which are disjoint to X for $s = g^i, 0 \leq i \leq |U|$ where g is a generator of the multiplicative group in GF(2^m).
  (b) Compute the determinants of $\mathbf{T}^{(s)}$.
  (c) Use Lagrange interpolation to solve for the M_i's via Eq. 4.3.
  (d) Calculate the $Z(i)$'s by Eq. 4.7.
  (e) Evaluate Eq. 4.4.

Given that the random U is such that all edges in *some* exact cover S are kept in H_U, the previous Section 4 verifies its correctness: Lemmas 4.2 and 4.3 together with the observation that g^i for $1 \leq i \leq |U|$ are all distinct points, assures us that step (3) of the algorithm works. We are left with deciding t and I to make it very likely that some exact solution is kept at least once and tune them to get the best possible runtime.

4.2. Runtime Analysis

Our runtime analysis hinges on the probability that any fixed solution S to the XkC instance $H = (V,E)$ when projected on a subset $U \subseteq V$ chosen uniformly at random from the tn-sized subsets of V for some fraction t of the vertices, gets all its edges to leave a small projection on U, namely $\forall e \in S : |e \cap U| \leq 2$. We denote this event by $\varepsilon(t)$. If we repeat the process I times, the probability that none of the I independent random selections for U is successful in the sense that they retain S after the projection, is at most $(1 - \Pr(\varepsilon(t)))^I < e^{-\Pr(\varepsilon(t))I}$. Consequently, we need $I = \log(\epsilon^{-1})\Pr(\varepsilon(t))^{-1}$ to get probability at least $1 - \epsilon$ for one or more of the I selections to be successful. Thus we may use $\epsilon = c^{-n}$ for some constant $c > 1$ to get an exponentially low probability in n of failure without increasing the number of repetitions I by more than a polynomial factor.

k	τ_{12}	τ_2	t	$I^{1/n}$	c_k
3	0.961	0.679	0.547	1.092	1.496
4	0.936	0.613	0.387	1.073	1.642
5	0.921	0.583	0.301	1.060	1.721
6	0.912	0.565	0.246	1.050	1.771
7	0.905	0.554	0.208	1.043	1.806
8	0.900	0.546	0.181	1.038	1.832

Table 2: Numerically found parameters τ_{12} and τ_2 which approximately minimizes c_k.

To bound the probability of the event, we count the number of good tn-sized subsets of the vertices. This is a binomial sum (actually a trinomial one) over the number of edges in the solution S which gets a projection of size two:

$$\sum_{t_1+2t_2=tn} \binom{n/k}{t_1}\binom{n/k-t_1}{t_2}\binom{k}{2}^{t_2}\binom{k}{1}^{t_1} \tag{4.8}$$

To lower bound this sum of all non-negative terms, we will use just one of them. Let $N = n/k$ and parametrize $tn = \tau_{12}N + \tau_2 N$ where τ_{12} is the fraction of sets in the solution S which gets at least one of its elements chosen, and τ_2 is the fraction of sets that gets two.

Then, we bound our probability as the quotient of the single term lower bound on the number of good sets and the number of all sets $\binom{n}{tn}$ to

$$\Pr(\varepsilon(t)) \geq \frac{\binom{N}{\tau_{12}N}\binom{\tau_{12}N}{\tau_2 N}k^{\tau_{12}N}(k-1)^{\tau_2 N}}{2^{\tau_2 N}\binom{kN}{(\tau_{12}+\tau_2)N}} \tag{4.9}$$

The runtime of Corollary 2.5 is $O^*(2^{n-tn})$ given our polynomial time algorithm for computing $W_{2,f}$. Omitting polynomial factors, Algorithm 4.4 for XkC has to run for $Pr(\varepsilon(t))^{-1}$ different choices of U in the worst case. Let $T_{k,t}$ denote the final runtime, and expand the binomials of Eq. 4.9 to get:

$$T_{k,t} \leq \frac{2^{n-tn}}{\Pr(\varepsilon(t))} \leq \frac{2^{kN-\tau_{12}N}(\tau_2 N)!(N-\tau_{12}N)!(\tau_{12}N-\tau_2 N)!(kN)!}{N!k^{\tau_{12}N}(k-1)^{\tau_2 N}(kN-\tau_{12}N-\tau_2 N)!(\tau_{12}N+\tau_2 N)!} \tag{4.10}$$

If we replace the factorials with Stirling's approximation $n! \in \theta(\sqrt{n}(n/e)^n)$ and divide $(N/e)^{k+1}$ out of both numerator and denominator, we are left with a slightly less intimidating expression

$$T_{k,t} \leq \left(\frac{2^{(k-\tau_{12})}\tau_2^{\tau_2}(\tau_{12}-\tau_2)^{\tau_{12}-\tau_2}(1-\tau_{12})^{1-\tau_{12}}}{k^{\tau_{12}-k}(k-1)^{\tau_2}(k-\tau_{12}-\tau_2)^{k-\tau_{12}-\tau_2}(\tau_{12}+\tau_2)^{\tau_{12}+\tau_2}}\right)^N \tag{4.11}$$

Rewriting this as $T_{k,t} \leq c_k^n$ we see that c_k can be obtained as the k:th root of the expression within the brackets in Eq. 4.11. Solving numerically for the choices of τ_{12} and τ_2 that minimizes c_k we find that the minimum moves slightly with increasing k, see Table 2. The minimum, however, lies in a quite flat neighborhood within a large vicinity of the actual minimum, and comparable bounds not too far from the best possible with our technique are obtained with fixed parameters for all k by, say, $\tau_{12} = 0.9$ and $\tau_2 = 0.6$. With this choice of parameters in Eq. 4.11 we obtain the general bound in Theorem 1.2.

Acknowledgements

This research was supported in part by the Swedish Research Council project ”Exact Algorithms”.

References

[1] A. Björklund and T. Husfeldt. Exact Algorithms for Exact Satisfiability and Number of Perfect Matchings. Algorithmica 52(2): 226-249, 2008.
[2] A. Björklund, T. Husfeldt, and M. Koivisto. Set Partitioning via inclusion–exclusion. SIAM Journal on Computing Vol.39, No.2: 546-563, 2009.
[3] J. R. Bunch and J. E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication, Mathematics of Computation, 28: 231236, 1974.
[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation, 9:251-280, 1990.
[5] J. Edmonds. Systems of distinct representatives and linear algebra. Journal of Research of the Natural Bureau of Standards, 71B, 4:241–245, 1967.
[6] R. Karp. Reducibility Among Combinatorial Problems. Complexity of Computer Computations. New York: Plenum. pp. 85-103, 1972.
[7] D. E. Knuth. Dancing Links, arXiv: cs/0011047, 2000.
[8] M. Koivisto. Partitioning into Sets of Bounded Cardinality. Proceedings of the 7th IWPEC, 2009.
[9] I. Koutis. Faster Algebraic Algorithms for Path and Packing Problems. 35th ICALP, pp. 575–586, 2008.
[10] L. Lovász. On determinants, matchings and random algorithms. Fundamentals of Computing Theory. Akademia-Verlag, Berlin, 1979.
[11] R. Motwani and P. Raghavan. Randomized algorithms. Cambridge University Press, 1995.
[12] H. J. Ryser. Combinatorial Mathematics. Carus Math. Monographs, no. 14. Math. Assoc. of America, Washington, DC, 1963.
[13] W. T. Tutte. The factorization of linear graphs. Journal of the London Mathematical Society, 22:107–111, 1947.
[14] L. G. Valiant. The Complexity of Computing the Permanent. Theor. Comput. Sci. 8: 189–201, 1979.
[15] R. Williams. Finding Paths of Length k in $O^*(2^k)$ Time. Information Processing Letters 109(6):315–318, 2009.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 107-118
www.stacs-conf.org

ON ITERATED DOMINANCE, MATRIX ELIMINATION, AND MATCHED PATHS

FELIX BRANDT [1] AND FELIX FISCHER [1] AND MARKUS HOLZER [2]

[1] Institut für Informatik, Ludwig-Maximilians-Universität München, 80538 München, Germany
E-mail address: {brandtf,fischerf}@tcs.ifi.lmu.de

[2] Institut für Informatik, Universität Gießen, 35392 Gießen, Germany
E-mail address: holzer@informatik.uni-giessen.de

Abstract. We study computational problems arising from the iterated removal of weakly dominated actions in anonymous games. Our main result shows that it is NP-complete to decide whether an anonymous game with three actions can be solved via iterated weak dominance. The two-action case can be reformulated as a natural elimination problem on a matrix, the complexity of which turns out to be surprisingly difficult to characterize and ultimately remains open. We however establish connections to a matching problem along paths in a directed graph, which is computationally hard in general but can also be used to identify tractable cases of matrix elimination. We finally identify different classes of anonymous games where iterated dominance is in P and NP-complete, respectively.

1. Introduction

An *anonymous game* is characterized by the fact that players do not distinguish between other players in the game, i.e., their payoff only depends on the number of other players playing the different actions, but not on their identities. Anonymous games constitute a very natural class of multi-player games which is also highly relevant in practice (cf. [7]). *Symmetric* games additionally have identical payoff functions for all players. A *strategy* of a player is a probability distribution over his actions, and we say that an action is *weakly dominated* if there exists a strategy of the same player guaranteeing him at least the same payoff for any combination of strategies of the other players, and strictly more payoff for some such combination.[1] Dominated actions may be discarded for the simple reason that the player will never face a situation where he would benefit from using these actions. The solution concept of *iterated dominance* works by removing a dominated action and applying the same reasoning to the reduced game (e.g., [15]). A game is then called *solvable* by iterated dominance if there is a sequence of eliminations that leaves only one action for

1998 ACM Subject Classification: F.2.2, J.4.

Key words and phrases: Algorithmic Game Theory, Computational Complexity, Iterated Dominance, Matching.

[1] Some authors (e.g., [10, 13]) use the terms weak dominance or dominance to refer to a weaker notion that does *not* require the dominating strategy to sometimes yield a strictly higher payoff. This notion is called very weak dominance by other authors (e.g., [2, 4]).

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE
27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2448

each player. Interestingly, anonymous games often arise in the context of voting, where dominance solvability was originally introduced [14].

Unlike iterated *strict* dominance, which requires the dominating action in each step to be strictly better for *every* combination of strategies of the other players, proper epistemic foundations for iterated *weak* dominance are fairly hard to come by (e.g., [3, 19]). Nevertheless, iterated weak dominance is an established and well-studied solution concept that occurs in virtually every textbook on game theory. Its computational properties, however, are not well understood, particularly in restricted classes like anonymous games. Potential computational hardness of iterated weak dominance stems from the fact that the result of the elimination process generally depends on the order in which actions are eliminated.

Related Work. Deciding whether a game in normal form can be solved by iterated weak dominance is NP-complete already for games with two players and two different payoffs and when restricted to dominance by pure strategies [10, 6]. In two-player constant-sum games, both solvability and eliminability of a given action become tractable, while reachability of a subgame remains NP-complete [4]. The corresponding problems for *strict* dominance can generally be solved in polynomial time [6].

All of the above results concern games with few players and an unbounded number of actions. Unlike general normal-form games, anonymous and symmetric games allow for a succinct representation even when the number of players is unbounded. Computational aspects of these games, particularly with respect to Nash equilibrium, have recently come under increased scrutiny due to their importance in modeling large anonymous environments like the Internet. A Nash equilibrium of a symmetric game can be found in polynomial time if the number of actions is not too large compared to the number of players [16]. In the larger class of anonymous games, Nash equilibria admit a polynomial-time approximation scheme when there is only a constant number of actions [7]. The pure equilibrium problem is tractable in anonymous games with a constant number of actions, and NP-complete if the number of actions grows in the number of players [5].

Results and Paper Structure. We begin by introducing the relevant game-theoretic concepts. In Section 3 we show that iterated dominance solvability is NP-hard for symmetric games with an unbounded number of actions, and tractable for symmetric games with a constant number of actions. The rest of the paper is then concerned with the only remaining class, anonymous games with a constant number of actions. In Section 4, we show how the two-action case can be reformulated as a natural elimination problem on a matrix. The complexity of this problem remains open, but in Section 5 we draw connections to a matching problem on paths of a directed graph. The latter problem, which may be of independent interest, is intractable in general but allows us to obtain efficient algorithms for restricted versions of matrix elimination. In Section 6 we finally use the matching formulation to show NP-hardness of iterated dominance in anonymous games with three actions. Proofs are omitted due to space constraints, and will be given in the full version of the paper.

2. Preliminaries

An accepted way to model situations of strategic interaction is by means of a *normal-form game* (e.g., [15]).

Definition 2.1 (normal-form game)**.** A *game in normal-form* is a tuple $\mathit{\Gamma} = (N, (A_i)_{i \in N}, (p_i)_{i \in N})$, where N is a finite set of *players* and for each player $i \in N$, A_i is a finite set of *actions* available to player i and $p_i : (\prod_{i \in N} A_i) \to \mathbb{R}$ is a function mapping each action profile, i.e., each combination of actions, to a real-valued *payoff* for player i.

We write $S_i = \mathit{\Delta}(A_i)$ for the set of (mixed) *strategies* of player $i \in N$, i.e., the set of probability distributions over his actions, and call a strategy *pure* if it selects some action with probability one. A vector $s \in \prod_{i \in N} S_i$ will be called a *strategy profile.* Payoff functions naturally extend to strategy profiles, and we write $p_i(s)$ for the *expected* payoff of player i in strategy profile s. We further write $n = |N|$ for the number of players in a game, s_i for the ith element of strategy profile s, and s_{-i} for the vector of all elements of s but s_i.

We will henceforth concentrate on games where $A_i = A$ for all $i \in N$ and some set A. Such a game is *anonymous* if the payoff of player i is invariant under any automorphism $\pi' : A^N \to A^N$ of the set of actions profiles induced by a permutation $\pi : N \to N$ of the set of players that satisfies $\pi(i) = i$ (e.g., [5]). An intuitive way to describe anonymous games is in terms of equivalence classes of the automorphism group of π', using a notion introduced by Parikh [18] in the context of context-free languages. Given a set A of actions, the *commutative image* of an action profile $a_N \in A^N$ is given by $\#(a_N) = (\#(a, a_N))_{a \in A}$ where $\#(a, a_N) = |\{\, i \in N : a_i = a \,\}|$. In other words, $\#(a, a_N)$ denotes the number of players playing action a in action profile a_N, and $\#(a_N)$ is the vector of these numbers for all the different actions. This definition naturally extends to action profiles for subsets of the players. We consider four types of anonymity (cf. [5]).

Definition 2.2 (anonymity)**.** Let $\mathit{\Gamma} = (N, (A_i)_{i \in N}, (p_i)_{i \in N})$ be a normal-form game, A a set of actions such that $A_i = A$ for all $i \in N$. $\mathit{\Gamma}$ is called

- *anonymous* if $p_i(a_N) = p_i(a'_N)$ for all $i \in N$ and all $a_N, a'_N \in A^N$ with $a_i = a'_i$ and $\#(a_{-i}) = \#(a'_{-i})$,
- *symmetric* if $p_i(a_N) = p_j(a'_N)$ for all $i, j \in N$ and all $a_N, a'_N \in A^N$ with $a_i = a'_j$ and $\#(a_{-i}) = \#(a'_{-j})$,
- *self-anonymous* if $p_i(a_N) = p_i(a'_N)$ for all $i \in N$ and all $a_N, a'_N \in A^N$ with $\#(a_N) = \#(a'_N)$, and
- *self-symmetric* if $p_i(a_N) = p_j(a'_N)$ for all $i, j \in N$ and all $a_N, a'_N \in A^N$ with $\#(a_N) = \#(a'_N)$.

When talking about anonymous games, we write $p_i(a_i, x_{-i})$ for the payoff of player i under any action profile a_N with $\#(a_{-i}) = x_{-i}$. For self-anonymous games, $p_i(x)$ is used to denote the payoff of player i under any profile a_N with $\#(a_N) = x$. Unless noted otherwise, we assume that anonymous games are given explicitly, i.e., as a list of payoffs for the different commutative images.

A well-known method for simplifying strategic games is the removal of actions that are weakly dominated by some strategy of the same player, in the sense that playing the latter is never worse than playing the former and sometimes strictly better. The removal of one or more dominated actions may render additional actions dominated, which may then iteratively be removed. To make these notions precise, we need some notation. Given a game $\mathit{\Gamma} = (N, (A_i)_{i \in N}, (p_i)_{i \in N})$, call an elimination sequence of $\mathit{\Gamma}$ a finite sequence $(D_1, D_2, \ldots, D_k)$ of subsets of the disjoint union of the sets A_i, i.e., $D_j \subseteq \cup_{i \in N} A^*_i$ for all j with $1 \leq j \leq k$, where $A^*_i = A_i \times \{i\}$. For a set $D \subseteq \cup_{i \in N} A^*_i$, denote by $\Gamma(D)$ the induced subgame of Γ where the actions in D have been removed, i.e., $\Gamma(D) = (N, (A'_i)_{i\in N}, (p_i|_{\prod_{i\in N} A'_i})_{i\in N})$ where $A'_i = \{\, a : (a,i) \in A^*_i \setminus D \,\}$.

Definition 2.3 (iterated dominance)**.** Let $\Gamma = (N, (A_i)_{i\in N}, (p_i)_{i\in N})$ be a game. An action $d_i \in A_i$ is said to be *(weakly) dominated* by strategy $s_i \in S_i$ if for all $b \in \prod_{j\in N} A_j$, $p_i(b_{-i}, d_i) \leq \sum_{a_i \in A_i} s_i(a_i) p_i(b_{-i}, a_i)$ and for at least one $b \in \prod_{j\in N} A_j$, $p_i(b_{-i}, d_i) < \sum_{a_i \in A_i} s_i(a_i) p_i(b_{-i}, a_i)$. An elimination sequence $(D_1, D_2, \ldots, D_m)$ of Γ is called *valid* if either it is the empty sequence, or if $(D_1, D_2, \ldots, D_{m-1})$ is valid in Γ and every $d_m \in D_m$ is dominated in $\Gamma(\cup_{1\leq j\leq m-1} D_j)$. An action $a \in \cup_{i\in N} A_i$ is called *eliminable* if there exists a valid elimination sequence $(D_1, D_2, \ldots, D_m)$ such that a is weakly dominated in $\Gamma(\cup_{1\leq j\leq m} D_j)$. Game Γ is called *solvable* if it is possible to obtain a game where only one action remains for each player, i.e., if there exists a valid elimination sequence $(D_1, D_2, \ldots, D_m)$ such that $\Gamma(\cup_{1\leq j\leq m} D_j) = (N, (A'_i)_{i\in N}, (p'_i)_{i\in N})$ with $|A'_i| = 1$ for all $i \in N$.

We call iterated dominance solvability (IDS) and eliminability (IDE) the computational problems that ask for solvability of a game and eliminability of a particular action. In contrast to iterated *strict* dominance, which requires the inequality to be strict for every action profile of the other players, the result of iterated weak dominance depends on the order in which actions are removed, since the elimination of an action may render actions of another player undominated (e.g., [2]).

Restricted types of iterated dominance can be obtained by requiring that the dominating strategy s_i is pure, or that the elements of an elimination sequence are singletons and actions thus have to be eliminated one at a time (e.g., [2]). As far as dominance by pure and mixed strategies is concerned, we will frequently exploit that the two versions coincide in games with two actions, and also in games with only two different payoffs [6]. All results hold for dominance by pure strategies *and* for dominance by mixed strategies. Valid elimination sequences consisting of singletons possess a somewhat less complicated structure. We therefore in some cases restrict our attention to this specialization, and refer to the corresponding computational problems as *stepwise* IDS and IDE. The results ultimately obtained for the two variants will be very similar. A different notion of *solvability* merely requires the remaining action profiles to yield a unique payoff to each of the players (e.g., [14]). We note, but do not show here, that all hardness and tractability results extend to this notion as well.

3. Complexity of Iterated Dominance

Intuitively, a large number of actions neutralizes the computational advantage obtained from anonymity, by allowing for a distinction of the players by means of the *actions they play*. The search for pure Nash equilibria, for example, is tractable for anonymous games with a constant number of actions, but becomes NP-hard as soon as the number of actions grows in the number of players [5]. In the latter case, the size of the explicit representation grows exponentially in the number of players, and one would expect natural instances of such games to be described succinctly (cf. [16]). While as a matter of fact the results of Brandt et al. [5] are established via a specific encoding of the payoff functions, namely Boolean circuits, they nevertheless provide interesting insights into the influence of restricted classes of payoff functions on the complexity of solving a game. We give a similar result for iterated dominance in self-symmetric games, hardness for the other classes follows by inclusion.

Theorem 3.1. *IDS and IDE are NP-hard for all four classes of anonymous games, even if the number of actions grows only logarithmically in the number of players, if only dominance by pure strategies is considered, and if there are only two different payoffs.*

In the case of symmetric games, iterated dominance becomes tractable when the number of actions is bounded by a constant.

Theorem 3.2. *For symmetric and self-symmetric games with a constant number of actions, IDS and IDE can be decided in polynomial time.*

In light of these two results, only one interesting class remains, namely anonymous games with a constant number of actions. To gain a better understanding of the problem, we restrict ourselves even further to games with two actions. It turns out that in this case iterated dominance can be reformulated in a natural way as an elimination problem on matrices. The latter is the topic of the following section.

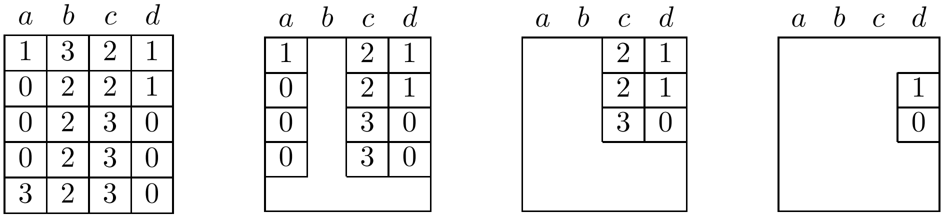
4. A Matrix Elimination Problem

Let $\mathit{\Gamma} = ([n], (\{0,1\})_{i\in N}, (p_i)_{i\in N})$ be a self-anonymous game with two actions for each player, and observe that the payoffs of $\mathit{\Gamma}$ can be represented by a matrix $X_{\mathit{\Gamma}} = (x_{i,j})_{(n+1)\times n}$ the ith row of which contains the payoff profile when exactly $i-1$ players play action 1, i.e., $x_{ij} = p_j(i-1)$. It will be instructive to view iterated dominance elimination in $\mathit{\Gamma}$ in terms of the corresponding operations on the matrix $X_{\mathit{\Gamma}}$. For now, we restrict our attention to the case where actions are eliminated one by one, and more generally consider matrices with an arbitrary number of rows and columns. It suffices to look at matrices whose entries are natural numbers.

Let X be an $m \times n$ matrix with entries from the natural numbers. Call a column c of X *increasing* for an interval I over the rows of X if the entries in c are monotonically increasing in I, with a strict increase somewhere in this interval. Analogously, call c *decreasing* for I if its entries are monotonically decreasing in I, with a strict decrease somewhere in this interval. Say that c is *active* for I if it is either increasing or decreasing for this interval. Now consider a process that starts with X and successively eliminates pairs of a row and a column. Rows will only be eliminated from the top or bottom, such that the remaining rows always form an interval over the rows of X. A column will only be eliminated if it is active for the remaining rows. Elimination of an increasing column is accompanied by elimination of the top row. Analogously, a decreasing column and the bottom row are eliminated at the same time. The process ends when no active columns remain.

Let us define the problem more formally. For a set A, $v \in A^n$, and $a \in A$, denote by $\#(a, v) = |\{\, \ell \leq n : v_\ell = a \,\}|$ the *commutative image* of a and v, and write $v_{\ldots k} = (c_1, c_2, \ldots, c_k)$ for the prefix of v of length $k \leq n$. Further denote $[n] = \{1, 2, \ldots, n\}$ and $[n]_0 = \{0, 1, \ldots, n\}$.

Definition 4.1 (matrix elimination)**.** Let $X \in \mathbb{N}^{m\times n}$ be a matrix. Call a column $k \in [n]$ of X *increasing* in an interval $[i,j] \subseteq [m]$ if the sequence $x_{ik}, x_{i+1,k}, \ldots, x_{jk}$ is monotonically increasing and $x_{ik} < x_{jk}$, *decreasing* in $[i,j] \subseteq [m]$ if $x_{ik}, x_{i+1,k}, \ldots, x_{jk}$ is monotonically decreasing and $x_{ik} > x_{jk}$, and *active* if it is either increasing or decreasing. Then, an *elimination sequence* of length k for X is a pair (c, r) such that $c \in [m]^k$, $r \in \{0,1\}^k$, and for all i, j with $1 \leq i < j \leq k$, $c_i \neq c_j$ and either $r_i = 0$ and column c_i is increasing in



$$\begin{array}{cccc} a & b & c & d \\ \hline 1 & 3 & 2 & 1 \\ 0 & 2 & 2 & 1 \\ 0 & 2 & 3 & 0 \\ 0 & 2 & 3 & 0 \\ 3 & 2 & 3 & 0 \end{array} \qquad \begin{array}{cccc} a & b & c & d \\ \hline 1 & & 2 & 1 \\ 0 & & 2 & 1 \\ 0 & & 3 & 0 \\ 0 & & 3 & 0 \\ & & & \end{array} \qquad \begin{array}{cccc} a & b & c & d \\ \hline & & 2 & 1 \\ & & 2 & 1 \\ & & 3 & 0 \\ & & & \\ & & & \end{array} \qquad \begin{array}{cccc} a & b & c & d \\ \hline & & & \\ & & & 1 \\ & & & 0 \\ & & & \\ & & & \end{array}$$

Figure 1: A matrix and a sequence of eliminations

$[\#(0, r_{\ldots i-1}) + 1, m - \#(1, r_{\ldots i-1})]$, or $r_i = 1$ and column c_i is decreasing in $[\#(0, r_{\ldots i-1}) + 1, m - \#(1, r_{\ldots i-1})]$.

Consider for example the sequence of matrices shown in Figure 1, obtained by starting with the 5×4 matrix on the left and successively eliminating columns b, a, c, and d. In this particular example, the process ends when all rows and columns of the matrix have been eliminated. If instead we eliminated columns c and a, no further eliminations would be possible. In fact, it would be obvious after the first elimination step that we cannot obtain a sequence of length 4: one of the columns not eliminated so far, column b, contains the same value in every row; this column cannot become active anymore, and, as a consequence, will never be eliminated.

What matters are not the actual matrix entries, but rather the difference between successive entries in a column. A more intuitive way to look at the problem may thus be in terms of a matrix with the number of rows reduced by one, and arrows pointing downward or upward if the value increases or decreases between two adjacent entries. A column can be deleted if it contains at least one arrow, and if all arrows in this column point in the same direction. The corresponding row to be deleted is the one at the base of the arrows.

We will be interested in two computational problems. Matrix elimination (ME) asks whether there exists an elimination sequence that deletes the whole matrix, i.e., one of length $\min(m-1, n)$. Eliminability of a column (CE) is given $k \in [n]$ and asks whether there exists an elimination sequence (c, r) such that for some i, $c_i = k$. Without restrictions on m and n, ME and CE turn out to be equivalent. Indeed, both of them are equivalent to the problem of deciding whether there exists an elimination sequence eliminating certain numbers of rows from the top and bottom of the matrix. Several other questions, like the one of an elimination sequence of a certain length, are equivalent as well.

Lemma 4.2. *CE and ME are equivalent under disjunctive truth-table reductions.*

When restricted to the case $m > n$, CE is at least as hard as ME in the sense that the latter can be reduced to the former while there is no obvious reduction in the other direction. The problem ME itself might be harder when the number of columns significantly exceeds the number of rows, because then the set of columns effectively needs to be partitioned into two sets of sizes m and $n - m$ of columns that have to be deleted and columns that can be discarded right away.

It is not hard to see that elimination of a matrix X is closely related to iterated dominance in the self-anonymous game described at the beginning of this section, where each player has two actions 0 and 1, and the payoff of player j when exactly $i - 1$ players play action 1 is given by matrix entry x_{ij}. Given actions for the other players, player j can choose between two adjacent entries of column j, so one of his two actions is dominated by the other one if the column is increasing or decreasing. Eliminating one of two actions effectively removes a player from the game, and elimination of the top or bottom row of the matrix mirrors the fact that the number of players who can still choose between both of their actions is reduced by one. Let us formally establish this relationship.

Lemma 4.3. *Stepwise* IDS *and* IDE *in anonymous games with two actions are equivalent under disjunctive truth-table reductions to* ME *and* CE*, respectively, restricted to instances with* $m = n + 1$.

We could have well allowed the simultaneous elimination of columns, and it is fairly obvious that the resulting computational problems would be equivalent to IDS and IDE. So why do we require columns to be eliminated one at a time? For one, solving ME and CE as defined above turns out to be intricate enough to begin with, and we will ultimately not be able to characterize their complexity. On the other hand, the additional structure afforded by stepwise elimination will help us to gain additional insights, which we will then use to prove the main result of this paper: NP-hardness of IDS and IDE in games with three actions, both for stepwise and simultaneous eliminations. Finally, much of the complexity of matrix elimination already appears to be present in the stepwise version, and any result for that version can probably be extended to simultaneous eliminations as well.

Solving ME in general turns out to be surprisingly complicated. A natural restriction can be obtained by requiring that all columns are increasing or decreasing in $[1, m]$. It is not too hard to show that this makes the problem tractable irrespective of the dimensions of the matrix, and we do so in the next section as a corollary of a slightly more general result. Unfortunately, tractability of this restricted case does not tell us a lot about the complexity of ME in general. The latter obviously becomes almost trivial if the order of elimination for the columns is known, i.e., if we are given $c \in [n]^k$ and ask whether there exists $r \in \{0, 1\}^k$ such that (c, r) is an elimination sequence. This observation directly implies membership in NP. More interestingly, deciding whether there exists $c \in [n]^k$ for a given $r \in \{0, 1\}^k$ such that (c, r) is an elimination sequence is also tractable. The reason is the specific "life cycle" of a column. Consider a matrix X, two intervals $I, J \subseteq [m]$ over the rows of X such that $J \subseteq I$, and a column $c \in [n]$ that is active in both I and J. Then, c must also be active for any interval K such that $J \subseteq K \subseteq I$, and c must either be increasing for all three intervals, or decreasing for all three intervals. Thus, r determines for every $i \in [k]$ a set of possible values for c_i, and leaves us with a matching problem in a bipartite graph with edges in $[n] \times [k]$. The latter can be solved in polynomial time. Closer inspection reveals that it can in fact be decomposed into two independent matching problems on convex bipartite graphs, for which the best known upper bound is NC^2 [11].

But what if nothing about c and r is known? Despite the fact that we can only eliminate the top or bottom row of the matrix in each step, this still amounts to an exponential number of possible sequences. The best upper bound for matching in convex bipartite graphs means that there currently is not much hope for constructing an algorithm that determines r nondeterministically and computes a matching on the fly. We can nevertheless use the above reasoning to recast the problem in the more general framework of *matching on paths.* For this, we will respectively identify intervals and pairs of intervals over the rows of X with vertices and edges of a directed graph G, and will then label each edge (I, J) by the identifiers of the columns of X that take I to J. An elimination sequence of length k for X then corresponds to a path of length k in G which starts at the vertex corresponding to the interval $[1, m]$, such that there exists a matching of size k between the edges on this path and the columns of X. In particular, by fixing a particular path, we obtain the bipartite matching problem described above. A more detailed discussion of this problem is the topic of the following section. We first study the problem itself, and return to matrix elimination toward the end of the section.

5. Matched Paths

The matching problem described in the previous section generalizes the well-studied class of matching problems between two disjoint sets, or bipartite matching problems, by requiring that the elements of one of the two sets form a certain sub-structure of a combinatorial structure. Most interesting from a computational perspective are variants where the underlying combinatorial structure can be identified in polynomial time, as it is the case for paths or for spanning trees.

Definition 5.1 (matching, matched path)**.** Let X be a set, Σ an alphabet, and $\sigma : X \to 2^{\Sigma}$ a labeling function assigning sets of labels to elements of X. Then, a *matching* of σ is a total function $f : X \to \Sigma$ such that for all $x, y \in X$, $f(x) \in \sigma(x)$ and $f(y) \neq f(x)$ if $y \neq x$.

Let $G = (V, E)$ be a directed graph, Σ an alphabet, and $\sigma : E \to 2^{\Sigma}$ a labeling function for edges of G. Then, a *matched path* of length k in G is a sequence $e_1, e_2, \dots, e_k$ such that for all i with $1 \leq i < k$, there exist $u, v, w \in V$ such that $e_i = (u, v)$ and $e_{i+1} = (v, w)$, and the restriction of σ to $\{ e_i : 1 \leq i \leq k \}$ has a matching.

We call matched path (MP) the computational problem that asks, for an explicitly given directed graph G with corresponding labeling function σ and an integer k, whether there exists a matched path of length k in G. Variants of this problem can be obtained by asking for a matching that contains a certain set of labels, or a matched path between a particular pair of vertices. These variants have an interesting interpretation in terms of sequencing with resources and multi-dimensional constraints on the utilization of these resources: every resource can be used in certain states corresponding to vertices of a directed graph, and their use causes transitions between states. The goal then is to find a sequence that uses a specific set or a certain number of resources, or one that reaches a certain state.

In the context of this paper, we are particularly interested in instances of MP corresponding to instances of ME. We will see later that the graphs of such instances are layered grid graphs (e.g., [1]), and that the labeling function satisfies a certain convexity property. But let us look at the general problem for a bit longer. Greenlaw et al. [12] consider the related *labeled graph accessibility problem*, which, given a directed graph G with a single label attached to each edge, asks whether there exists a path such that the concatenation of the labels along the path is a member of a context free language L given as part of the input. This problem is P-complete in general and LOGCFL-complete if G is acyclic. A matching, however, corresponds to a partial permutation of the members of the alphabet, and the number of nonterminal symbols of any context-free grammar in Chomsky normal form for the permutation language over Σ grows super-polynomially in the size of Σ [8]. It thus should not come as a surprise that the problem becomes harder when we ask for a matching. Indeed, MP bears some resemblance to the NP-complete problem *forbidden pairs* of finding a path in a directed or undirected graph if certain pairs of nodes or edges may not be used together [9]. Instead of reducing forbidden pairs to MP, however, we show NP-hardness of a restricted version of MP using a more complicated construction, on which we will be able to build in Section 6. To formally state the result we need some terminology.

Let $G = (V, E)$ be a directed graph with vertex set $V = [m]_0 \times [n]_0$. Call $(u, v) \in E$ a *south edge* if for some i and j, $u = (i, j)$ and $v = (i + 1, j)$, and an *east edge* if for some i and j, $u = (i, j)$ and $v = (i, j + 1)$. Then, G is called an $m \times n$ *layered grid graph* if it contains only south and east edges. In labeled graphs, nonexistent edges and edges that are mapped to the empty set by the labeling function are equivalent. We therefore concentrate on *complete* layered grid graphs, i.e., those containing all south and all east edges.

Theorem 5.2. *MP is NP-complete. Hardness holds even if G is a complete layered grid graph, $|\sigma(e)| \leq 1$ for every $e \in E$, and $|\{ e \in E : \lambda \in \sigma(e) \}| \leq 2$ for every $\lambda \in \Sigma$.*

The proof of this theorem starts by looking at a complete $m \times n$ grid graph G for appropriate values of m and n, and at a labeling function $\sigma : [m]_0 \cup [n]_0 \to \Sigma$. The latter can be interpreted as a labeling function for edges of G where a label either appears on all the edges in a given row or column or on none of them. Labels in Σ correspond to variable occurrences in an instance of the NP-complete problem *balanced one-in-three 3SAT* [17], and σ is defined in such a way that a path through the graph corresponds to an assignment of truth values to variable occurrences. The overall structure of the graph consist of two parts. In the first part, consistency of the overall assignment is ensured by placing labels corresponding to different occurrences of the same variable on the same path. In the second part, the same labels are used again to verify that all clauses are satisfied by the assignment. To obtain Theorem 5.2 and get a better understanding of the minimal requirements for hardness, the graph is then modified further. An important property of the labeling function in this context seems to be that the same label can appear at least twice in different parts of the graph.

The labeling function σ can also more generally be interpreted as belonging to a more general graph where transitions can take place from any vertex to any other vertex to the south and east of it, as long as the distance in columns between the two vertices is at most the number of unused labels that appear on the row associated with the former vertex, and the same condition holds for the distance in rows and the number of labels on the column. Intuitively, this type of transition occurs when several dominated actions of a game are eliminated simultaneously. It will play an important role in the proof of Theorem 6.1.

Let us now return to matrix elimination. In light of Theorem 5.2, an efficient algorithm for ME would have to exploit additional structure of MP instances induced by instances of ME. This structure is indeed quite restricted in that edges carrying a particular label λ satisfy a "directed" convexity condition: if λ appears on two edges $e = (u, v)$ and $e' = (u', v')$, then λ must appear on *all* south edges or on all east edges that lie on a path from u to v', but not both. In particular, if there is such a path, it cannot be that one of e and e' is a south edge and the other is an east edge. This fact is illustrated in Figure 2, which shows the labeled graph for the ME instance of Figure 1, as well as a matched path corresponding to an elimination sequence of maximum length.

Definition 5.3 (directed convexity)**.** Let $G = (V, E)$ be a complete layered grid graph. A labeling function $\sigma : E \to 2^{\Sigma}$ for G is called *directed convex* if for every label $\lambda \in \Sigma$ and for every set of three edges $e_1 = (u_1, v_1)$, $e_2 = (u_2, v_2)$, $e_3 = (u_3, v_3)$, such that u_2 is reachable from u_1, u_3 is reachable from u_2, and $\lambda \in \sigma(e_1) \cap \sigma(e_3)$, it holds that e_1 and e_3 have the same direction and $\lambda \in \sigma(e_2)$ if and only if e_2 has the same direction as well.

It is not too hard to see that instances corresponding to ME have a directed convex labeling function.

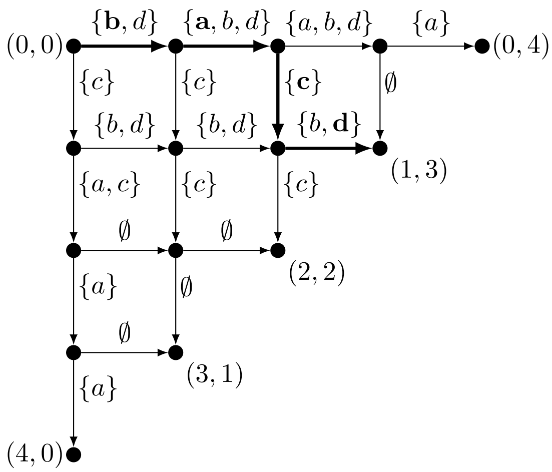


Figure 2: Labeled graph for the matrix elimination instance of Figure 1. A matched path and its matching are shown in bold.

Lemma 5.4. *ME is polynomial time many-one reducible to MP restricted to layered grid graphs and directed convex labeling functions.*

Directed convexity of the labeling function means that we cannot show NP-hardness of ME by a construction similar to the one used in the proof of Theorem 5.2. On the other hand, it is not quite clear how the additional structure provided by directed convexity can be exploited to obtain a polynomial-time algorithm for ME. The case $m \leq n$ will probably add additional complications. We therefore leave the complexity of ME as an open problem, albeit quite an elegant one.

Here we consider a more special case of MP, which provides additional insights. In the corresponding instances of ME, all columns are active at the beginning of the matrix elimination process, or all columns are active in the interval of length one at the end of the elimination process.

Definition 5.5 (backward and forward closure)**.** Let $G = (V, E)$ be a complete layered grid graph. Let s be the unique vertex of G with in-degree zero, t the unique vertex with outdegree zero. Then, a labeling function $\sigma : E \to 2^{\Sigma}$ for G is called *backward closed* if $\{\lambda \in \sigma(s, v) : (s, v) \in E\} = \Sigma$. Similarly, σ is called *forward closed* if $\{\lambda \in \sigma(s, v) : (v, t) \in E\} = \Sigma$.

It may not have gone unnoticed that these properties are closely related to closure properties found respectively in matroids and antimatroids. Together with directed convexity, each of the closure properties further implies that each label appears only on east edges or only on south edges. This allows us to consider two distinct matching problems, one for east and one for south edges, and obtain a tractability result.

Theorem 5.6. *Let $G = (V, E)$ be a complete layered grid graph, σ a labeling function for G that is directed convex and either backward or forward closed. Then, MP for G and σ can be solved in nondeterministic logarithmic space.*

A generalization of both backward and forward closure can be obtained by considering labeling functions that are *connected* in the sense that the edges carrying a particular

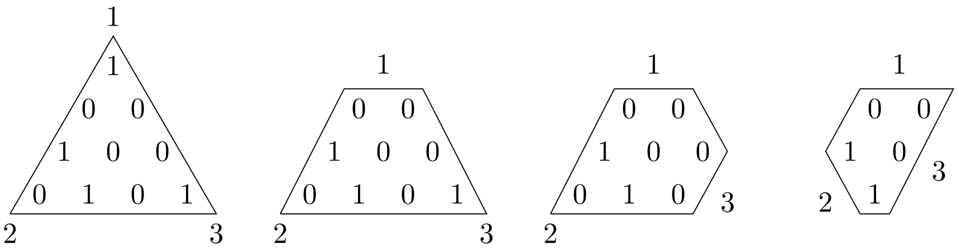


Figure 3: Payoffs of a particular player in a self-anonymous game with $n = 3$ and $k = 3$. Initially all actions are pairwise undominated. If one of the other players eliminates action 1, action 3 weakly dominates action 1. Action 1 then becomes undominated if some player deletes action 3, and dominated by action 2 if one more player deletes action 3, and some player deletes action 2.

label, together with all edges in the respective other direction, form a weakly connected graph. This property introduces a dependence between the matching problems for the two directions, and a very interesting question is whether Theorem 5.6 can be generalized to this setting.

6. Self-Anonymous Games with a Constant Number of Actions

It is natural to ask whether iterated dominance for games with more than two actions can still be interpreted in terms of eliminations in a matrix or matrix-like structure. Consider a self-anonymous game with k actions. As before, the payoff of a particular player i only depends on the number of players, including the player itself, that play each of the different actions. They can thus be written down as entries in a discrete simplex of dimension $k - 1$. The elimination of the ℓth action by some player can then be interpreted as a cut along the ℓth 0-face of the simplex of every player.

The left hand side of Figure 3 shows the payoffs of a particular player in a self-anonymous game with $n = 3$ and $k = 3$. Compared to matrix elimination as introduced in Definition 4.1 and illustrated in Figure 1, we notice an interesting shift, which curiously has nothing to do with the added possibility of dominance by mixed strategies. Rather, a particular action $a \in A$ may now be eliminated by either one of several other actions in $A \setminus \{a\}$, and the situations where a can be eliminated no longer form a convex set.

This already indicates that it might be possible to construct a layered grid graph with corresponding labeling function for which the existence of a matched path is NP-hard to decide, and which is induced by a self-anonymous game with three actions for each player. To obtain our main result we however have to overcome one additional obstacle: when dropping the assumption that actions are eliminated one at a time, the equivalence between elimination sequences and labeled paths in a layered grid graph breaks down. We therefore start from the construction used in the proof of Theorem 5.2, and use additional vertices and labels to make it work for the more general type of transitions corresponding to the simultaneous elimination of actions.

Theorem 6.1. *IDS and IDE are NP-complete. Hardness holds even for self-anonymous games with three actions and two different payoffs, and also applies to stepwise IDS and IDE.*

Acknowledgements

This material is based upon work supported by the Deutsche Forschungsgemeinschaft under grants BR 2312/3-1 and BR 2312/3-2. We thank Hermann Gruber, Paul Harrenstein, Tim Roughgarden, Inbal Talgam, and Michael Tautschnig for valuable discussions, and apologize to Edith Hemaspaandra for spoiling the sunset at White Sands.

References

[1] E. Allender, D. A. Mix Barrington, T. Chakraborty, S. Datta, and S. Roy. Grid graph reachability problems. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC)*, pages 299–313, 2006.

[2] K. R. Apt. Uniform proofs of order independence for various strategy elimination procedures. *Contributions to Theoretical Economics*, 4(1), 2004.

[3] A. Brandenburger, A. Friedenberg, and H. J. Keisler. Admissibility in games. *Econometrica*, 76(2): 307–352, 2008.

[4] F. Brandt, M. Brill, F. Fischer, and P. Harrenstein. On the complexity of iterated weak dominance in constant-sum games. In M. Mavronicolas and V. G. Papadopoulou, editors, *Proceedings of the 2nd International Symposium on Algorithmic Game Theory (SAGT)*, volume 5814 of *Lecture Notes in Computer Science (LNCS)*, pages 287–298. Springer-Verlag, 2009.

[5] F. Brandt, F. Fischer, and M. Holzer. Symmetries and the complexity of pure Nash equilibrium. *Journal of Computer and System Sciences*, 75(3):163–177, 2009.

[6] V. Conitzer and T. Sandholm. Complexity of (iterated) dominance. In *Proceedings of the 6th ACM Conference on Electronic Commerce (ACM-EC)*, pages 88–97. ACM Press, 2005.

[7] C. Daskalakis and C. H. Papadimitriou. Discretized multinomial distributions and Nash equilibria in anonymous games. In *Proceedings of the 49th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, 2008.

[8] K. Ellul, B. Krawetz, J. Shallit, and M.-W. Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 9(2–3):233–256, 2004.

[9] H. N. Gabow, S. N. Maheshwari, and L. Osterweil. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, 2(3):227–231, 1976.

[10] I. Gilboa, E. Kalai, and E. Zemel. The complexity of eliminating dominated strategies. *Mathematics of Operations Research*, 18(3):553–565, 1993.

[11] F. Glover. Maximum matching in convex bipartite graphs. *Naval Research Logistics Quarterly*, 14: 313–316, 1967.

[12] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation.* Oxford University Press, 1995.

[13] D. E. Knuth, C. H. Papadimitriou, and J. N. Tsitsiklis. A note on strategy elimination in bimatrix games. *Operations Research Letters*, 7:103–107, 1988.

[14] H. Moulin. Dominance solvable voting schemes. *Econometrica*, 47:1337–1351, 1979.

[15] R. B. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, 1991.

[16] C. H. Papadimitriou and T. Roughgarden. Computing equilibria in multi-player games. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 82–91. SIAM, 2005.

[17] I. Parberry. On the computational complexity of optimal sorting network verification. In *Proceedings of the Conference on Parallel Architectures and Languages Europe (PARLE)*, volume 505 of *Lecture Notes in Computer Science (LNCS)*, pages 252–269. Springer-Verlag, 1991.

[18] R. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

[19] L. Samuelson. Dominated strategies and common knowledge. *Games and Economic Behavior*, 4:284–313, 1992.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 119-130
www.stacs-conf.org

AMS WITHOUT 4-WISE INDEPENDENCE ON PRODUCT DOMAINS

VLADIMIR BRAVERMAN[1] AND KAI-MIN CHUNG[2] AND ZHENMING LIU[3] AND MICHAEL MITZENMACHER[4] AND RAFAIL OSTROVSKY[5]

[1] University of California Los Angeles. Supported in part by NSF grants 0716835, 0716389, 0830803, 0916574 and Lockheed Martin Corporation.
E-mail address: vova@cs.ucla.edu
URL: http://www.cs.ucla.edu/~vova

[2] Harvard School of Engineering and Applied Sciences. Supported by US-Israel BSF grant 2006060 and NSF grant CNS-0831289.
E-mail address: kmchung@fas.harvard.edu
URL: http://people.seas.harvard.edu/~kmchung/

[3] Harvard School of Engineering and Applied Sciences. Supported in part by NSF grant CNS-0721491. The work was finished during an internship in Microsoft Research Asia.
E-mail address: zliu@fas.harvard.edu
URL: http://people.seas.harvard.edu/~zliu/

[4] Harvard School of Engineering and Applied Sciences. Supported in part by NSF grant CNS-0721491 and research grants from Yahoo!, Google, and Cisco.
E-mail address: michaelm@eecs.harvard.edu
URL: http://www.eecs.harvard.edu/~michaelm/

[5] University of California Los Angeles. Supported in part by IBM Faculty Award, Lockheed-Martin Corporation Research Award, Xerox Innovation Group Award, the Okawa Foundation Award, Intel, Teradata, NSF grants 0716835, 0716389, 0830803, 0916574 and U.C. MICRO grant.
E-mail address: rafail@cs.ucla.edu
URL: http://www.cs.ucla.edu/~rafail

ABSTRACT. In their seminal work, Alon, Matias, and Szegedy introduced several sketching techniques, including showing that 4-wise independence is sufficient to obtain good approximations of the second frequency moment. In this work, we show that their sketching technique can be extended to product domains $[n]^k$ by using the product of 4-wise independent functions on $[n]$. Our work extends that of Indyk and McGregor, who showed the result for $k = 2$. Their primary motivation was the problem of identifying correlations in data streams. In their model, a stream of pairs $(i, j) \in [n]^2$ arrive, giving a joint distribution (X, Y), and they find approximation algorithms for how close the joint distribution is to the product of the marginal distributions under various metrics, which naturally corresponds to how close X and Y are to being independent. By using our technique, we obtain a new result for the problem of approximating the ℓ_2 distance between the joint distribution and the product of the marginal distributions for k-ary vectors, instead of just pairs, in a single pass. Our analysis gives a randomized algorithm that is a $(1 \pm \epsilon)$ approximation (with probability $1 - \delta$) that requires space logarithmic in n and m and proportional to 3^k.

1998 ACM Subject Classification: F.2.1, G.3 .
Key words and phrases: Data Streams, Randomized Algorithms, Streaming Algorithms, Independence, Sketches.

THIS PAPER IS A MERGE FROM THE WORK OF [7, 9, 10]

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2449

1. Introduction

In their seminal work, Alon, Matias and Szegedy [4] presented celebrated sketching techniques and showed that 4-wise independence is sufficient to obtain good approximations of the second frequency moment. Indyk and McGregor [12] make use of this technique in their work introduce the problem of measuring independence in the streaming model. There they give efficient algorithms for approximating pairwise independence for the ℓ_1 and ℓ_2 norms. In their model, a stream of pairs $(i, j) \in [n]^2$ arrive, giving a joint distribution (X, Y), and the notion of approximating pairwise independence corresponds to approximating the distance between the joint distribution and the product of the marginal distributions for the pairs. Indyk and McGregor state, as an explicit open question in their paper, the problem of whether one can estimate k-wise independence on k-tuples for any $k > 2$. In particular, Indyk and McGregor show that, for the ℓ_2 norm, they can make use of the product of 4-wise independent functions on $[n]$ in the sketching method of Alon, Matias, and Szegedy. We extend their approach to show that on the product domain $[n]^k$, the sketching method of Alon, Matias, and Szegedy works when using the product of k copies of 4-wise independent functions on $[n]$. The cost is that the memory requirements of our approach grow exponentially with k, proportionally to 3^k.

Measuring independence and k-wise independence is a fundamental problem with many applications (see e.g., Lehmann [13]). Recently, this problem was also addressed in other models by, among others, Alon, Andoni, Kaufman, Matulef, Rubinfeld and Xie [1]; Batu, Fortnow, Fischer, Kumar, Rubinfeld and White [5]; Goldreich and Ron [11]; Batu, Kumar and Rubinfeld [6]; Alon, Goldreich and Mansour [3]; and Rubinfeld and Servedio [15]. Traditional non-parametric methods of testing independence over empirical data usually require space complexity that is polynomial to either the support size or input size. The scale of contemporary data sets often prohibits such space complexity. It is therefore natural to ask whether we will be able to design algorithms to test for independence in streaming model. Interestingly, this specific problem appears not to have been introduced until the work of Indyk and McGregor. While arguably results for the ℓ_1 norm would be stronger than for the ℓ_2 norm in this setting, the problem for ℓ_2 norms is interesting in its own right. The problem for the ℓ_1 norm has been recently resolved by Braverman and Ostrovsky in [8]. They gave an $(1 \pm \epsilon, \delta)$-approximation algorithm that makes a single pass over a data stream and uses polylogarithmic memory.

1.1. Our Results

In this paper we generalize the "sketching of sketches" result of Indyk and McGregor. Our specific theoretical contributions can be summarized as follows:

Main Theorem.
Let $\vec{v} \in \mathrm{R}^{(n^k)}$ be a vector with entries $\vec{v}_{\mathbf{p}} \in R$ for $\mathbf{p} \in [n]^k$. Let $h_1, \ldots, h_k : [n] \to \{-1, 1\}$ be independent copies of 4-wise independent hash functions; that is, $h_i(1), \ldots, h_i(n) \in \{-1, 1\}$ are 4-wise independent hash functions for each $i \in [k]$, and $h_1(\cdot), \ldots, h_k(\cdot)$ are mutually independent. Define $H(\mathbf{p}) = \prod_{i=1}^{k} h_j(p_j)$, and the sketch $Y = \sum_{p \in [n]^k} \vec{v}_{\mathbf{p}} H(p)$.

We prove that the sketch Y can be used to give an efficient approximation for $\|\vec{v}\|^2$; our result is stated formally in Theorem 4.2. Note that H is not 4-wise independent.

As a corollary, the main application of our main theorem is to extend the result of Indyk and McGregor [12] to detect the dependency of k random variables in streaming model.

Corollary 1.1. *For every $\epsilon > 0$ and $\delta > 0$, there exists a randomized algorithm that computes, given a sequence $a_1, \ldots, a_m$ of k-tuples, in one pass and using $O(3^k \epsilon^{-2} \log \frac{1}{\delta}(\log m + \log n))$ memory bits, a number Y so that the probability Y deviates from the ℓ_2 distance between product and joint distribution by more than a factor of $(1 + \epsilon)$ is at most δ.*

1.2. Techniques and a Historical Remark

This paper is merge from [7, 9, 10], where the same result was obtained with different proofs. The proof of [10] generalizes the geometric approach of Indyk and McGregor [12] with new geometric observations. The proofs of [7, 9] are more combinatorial in nature. These papers offer new insights, but due to the space limitation, we focus on the proof from [9] in this paper. Original papers are available on line and are recommended to the interested reader.

2. The Model

We provide the general underlying model. Here we mostly follow the notation of [7, 12].

Let S be a stream of size m with elements $a_1, \ldots, a_m$, where $a_i \equiv (a_i^1, \ldots, a_i^k) \in [n]^k$. (When we have a sequence of elements that are themselves vectors, we denote the sequence number by a subscript and the vector entry by a superscript when both are needed.) The stream S defines an *empirical* distribution over $[n]^k$ as follows: the frequency $f(\omega)$ of an element $\omega \in [n]^k$ is defined as the number of times it appears in S, and the empirical distribution is

$$\Pr[\omega] = \frac{f(\omega)}{m} \quad \text{for any } \omega \in [n]^k.$$

Since $\omega = (\omega_1, \ldots, \omega_k)$ is a vector of size k, we may also view the streaming data as defining a joint distribution over the random variables $X_1, \ldots, X_k$ corresponding to the values in each dimension. (In the case of $k = 2$, we write the random variables as X and Y rather than X_1 and X_2.) There is a natural way of defining marginal distribution for the random variable X_i: for $\omega_i \in [n]$, let $f_i(\omega_i)$ be the number of times ω_i appears in the ith coordinate of an element of S, or

$$f_i(\omega_i) = \left|\{a_j \in S : a_j^i = \omega_i\}\right|.$$

The empirical marginal distribution $\Pr_i[\cdot]$ for the ith coordinate is defined as

$$\Pr_i[\omega_i] = \frac{f_i(\omega_i)}{m} \quad \text{for any } \omega_i \in [n].$$

Next let $\vec{v}$ be the vector in $\mathrm{R}^{[n]^k}$ with $\vec{v}_\omega = \Pr[\omega] - \prod_{1 \le i \le k} \Pr_i[\omega_i]$ for all $\omega \in [n]^k$. Our goal is to approximate the value

$$\|\vec{v}\| \equiv \left(\sum_{\omega \in [n]^k} \left| \Pr[\omega] - \prod_{1 \le i \le k} \Pr_i[\omega_i] \right|^2 \right)^{\frac{1}{2}}. \tag{2.1}$$

This represent the ℓ_2 norm between the tensor of the marginal distributions and the joint distribution, which we would expect to be close to zero in the case where the X_i were truly independent.

Finally, our algorithms will assume the availability of 4-wise independent hash functions. For more on 4-wise independence, including efficient implementations, see [2, 16]. For the purposes of this paper, the following simple definition will suffice.

Definition 2.1. *(4-wise independence)* A family of hash functions $\mathcal{H}$ with domain $[n]$ and range $\{-1, 1\}$ is *4-wise independent* if for any distinct values $i_1, i_2, i_3, i_4 \in [n]$ and any $b_1, b_2, b_3, b_4 \in \{-1, 1\}$, the following equality holds,

$$\Pr_{h \leftarrow \mathcal{H}}[h(i_1) = b_1, h(i_2) = b_2, h(i_3) = b_3, h(i_4) = b_4] = 1/16.$$

Remark 2.2. In [12], the family of 4-wise independent hash functions $\mathcal{H}$ is called 4-wise independent random vectors. For consistencies within our paper, we will always view the object $\mathcal{H}$ as a hash function family.

3. The Algorithm and its Analysis for $k = 2$

We begin by reviewing the approximation algorithm and associated proof for the ℓ_2 norm given in [12]. Reviewing this result will allow us to provide the necessary notation and frame the setting for our extension to general k. Moreover, in our proof, we find that a constant in Lemma 3.1 from [12] that we subsequently generalize appears incorrect. (Because of this, our proof is slightly different and more detailed than the original.) Although the error is minor in the context of their paper (it only affects the constant factor in the order notation), it becomes more important when considering the proper generalization to larger k, and hence it is useful to correct here.

In the case $k = 2$, we assume that the sequence $(a_1^1, a_1^2), (a_2^1, a_2^2), \ldots, (a_m^1, a_m^2)$ arrives an item by an item. Each (a_i^1, a_i^2) (for $1 \leq i \leq m$) is an element in $[n]^2$. The random variables X and Y over $[n]$ can be expressed as follows:

$$\left\{ \begin{array}{lclcl} \Pr[i,j] & = & \Pr[X = i, Y = j] & = & |\{\ell : (a_\ell^1, a_\ell^2) = (i,j)\}|/m \\ \Pr_1[i] & = & \Pr[X = i] & = & |\{\ell : (a_\ell^1, a_\ell^2) = (i, \cdot)\}|/m \\ \Pr_2[j] & = & \Pr[Y = j] & = & |\{\ell : (a_\ell^1, a_\ell^2) = (\cdot, j)\}|/m. \end{array} \right.$$

We simplify the notation and use $p_i \equiv \Pr[X = i]$, $q_j \equiv \Pr[Y = j]$, $r_{i,j} = \Pr[X = i, Y = j]$. and $s_{i,j} = \Pr[X = i]\Pr[Y = j]$.

Indyk and McGregor's algorithm proceeds in a similar fashion to the streaming algorithm presented in [4]. Specifically let $s_1 = 72\epsilon^{-2}$ and $s_2 = 2\log(1/\delta)$. The algorithm computes s_2 random variables $Y_1, Y_2, \ldots, Y_{s_2}$ and outputs their median. The output is the algorithm's estimate on the norm of v defined in Equation 2.1. Each Y_i is the average of s_1 random variables Y_{ij}: $1 \leq j \leq s_1$, where Y_{ij} are independent, identically distributed random variables. Each of the variables $D = D_{ij}$ can be computed from the algorithmic routine shown in Figure 1.

```
2-D APPROXIMATION $((a_1^1, a_1^2), \ldots, (a_m^1, a_m^2))$
Independently generate 4-wise independent random functions $h_1, h_2$ from $[n]$ to $\{-1, 1\}$.
for $c \leftarrow 1$ to $m$
    do Let the $c$th item $(a_c^1, a_c^2) = (i, j)$
        $t_1 \leftarrow t_1 + h_1(i)h_2(j)$, $t_2 \leftarrow t_2 + h_1(i)$, $t_3 \leftarrow t_3 + h_2(j)$.
Return $Y = (t_1/m - t_2 t_3/m^2)^2$.
```

Figure 1: The procedure for generating random variable Y for $k = 2$.

By the end of the process 2-D APPROXIMATION, we have $t_1/m = \sum_{i,j \in [n]} h_1(i)h_2(j)r_{i,j}$, $t_2/m = \sum_{i \in [n]} h_1(i)p_i$, and $t_3/m = \sum_{i \in [n]} h_2(i)q_i$. Also, when a vector is in $\mathrm{R}^{(n^2)}$, its indices can be represented by $(i_1, i_2) \in [n]^2$. In what follows, we will use a bold letter to represent the index of a high dimensional vector, e.g., $v_{\mathbf{i}} \equiv v_{i_1,i_2}$. The following Lemma shows that the expectation of Y is $\|v\|^2$ and the variance of Y is at most $8(\mathrm{E}[Y])^2$ because $\mathrm{E}[Y^2] \leq 9\mathrm{E}[Y]^2$.

Lemma 3.1. *(*[12]*) Let h_1, h_2 be two independent instances of 4-wise independent hash functions from $[n]$ to $\{-1,1\}$. Let $v \in \mathrm{R}^{n^2}$ and $H(\mathbf{i})(\equiv H\big((i_1,i_2)\big) = h_1(i_i) \cdot h_2(i_2)$. Let us define $Y = \Big(\sum_{\mathbf{i}\in[n]^2} H(\mathbf{i})v_{\mathbf{i}}\Big)^2$. Then $\mathrm{E}[Y] = \sum_{\mathbf{i}\in[n]^2} \vec{v}_{\mathbf{i}}^2$ and $\mathrm{E}[Y^2] \leq 9(\mathrm{E}[Y])^2$, which implies $\mathrm{Var}[Y] \leq 8E^2[Y]$.*

Proof. We have $\mathrm{E}[Y] = \mathrm{E}[(\sum_{\mathbf{i}} H(\mathbf{i})\vec{v}_{\mathbf{i}})^2] = \sum_{\mathbf{i}} \vec{v}_{\mathbf{i}}^2 \mathrm{E}[H^2(\mathbf{i})] + \sum_{\mathbf{i}\neq\mathbf{j}} \vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\mathrm{E}[H(\mathbf{i})H(\mathbf{j})]$. For all $\mathbf{i} \in [n]^2$, we know $h^2(\mathbf{i}) = 1$. On the other hand, $H(\mathbf{i})H(\mathbf{j}) \in \{-1,1\}$. The probability that $H(\mathbf{i})H(\mathbf{j}) = 1$ is $\Pr[H(\mathbf{i})H(\mathbf{j}) = 1] = \Pr[h_1(i_1)h_1(j_1)h_2(i_2)h_2(j_2) = 1] = 1/16 + \binom{4}{2}1/16 + 1/16 = 1/2$. The last equality holds is because $h_1(i_1)h_1(j_1)h_2(i_2)h_2(j_2) = 1$ is equivalent to saying either all these variables are 1, or exactly two of these variables are -1, or all these variables are -1. Therefore, $\mathrm{E}[h(\mathbf{i})h(\mathbf{j})] = 0$. Consequently, $\mathrm{E}[Y] = \sum_{\mathrm{i}\in[n]^2}(\vec{v}_{\mathrm{i}})^2$.

Now we bound the variance. Recall that $\mathrm{Var}[Y] = \mathrm{E}[Y^2] - \mathrm{E}[Y]^2$, we bound

$$\mathrm{E}[Y^2] = \sum_{\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}\in[n]^2} \mathrm{E}[H(\mathbf{i})H(\mathbf{j})H(\mathbf{k})h(\mathbf{l})]\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}} \leq \sum_{\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}\in[n]^2} |\mathrm{E}[H(\mathbf{i})H(\mathbf{j})H(\mathbf{k})H(\mathbf{l})]|\cdot|\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}}|.$$

Also $|\mathrm{E}[H(\mathbf{i})H(\mathbf{j})H(\mathbf{k})H(\mathbf{l})]| \in \{0,1\}$. The quantity $\mathrm{E}[H(\mathbf{i})H(\mathbf{j})H(\mathbf{k})H(\mathbf{l})] \neq 0$ if and only if the following relation holds,

$$\forall s \in [2] : ((i_s = j_s) \wedge (k_s = l_s)) \vee ((i_s = k_s) \wedge (j_s = l_s)) \vee ((i_s = l_s) \wedge (k_s = j_s))\,. \tag{3.1}$$

Denote the set of 4-tuples $(\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l})$ that satisfy the above relation by $\mathcal{D}$. We may also view each 4-tuple as an ordered set that consists of 4 points in $[n]^2$. Consider the unique smallest axes-parallel rectangle in $[n]^2$ that contains a given 4-tuple in $\mathcal{D}$ (i.e. contains the four ordered points). Note this could either be a (degenerate) line segment or a (non-degenerate) rectangle, as we discuss below. Let $M : \mathcal{D} \to \{A,B,C,D\}$ be the function that maps an element $\sigma \in \mathcal{D}$ to the smallest rectangle $ABCD$ defined by σ. Since a rectangle can be uniquely determined by its diagonals, we may write $M : \mathcal{D} \to (\chi_1,\chi_2,\varphi_1,\varphi_2)$, where $\chi_1 \leq \chi_2 \in [n]$, $\varphi_1 \leq \varphi_2 \in [n]$ and the corresponding rectangle is understood to be the one with diagonal $\{(\chi_1,\varphi_1),(\chi_2,\varphi_2)\}$. Also, the inverse function $M^{-1}(\chi_1,\chi_2,\varphi_1,\varphi_2)$ represents the pre-images of $(\chi_1,\chi_2,\varphi_1,\varphi_2)$ in $\mathcal{D}$. $(\chi_1,\chi_2,\varphi_1,\varphi_2)$ is degenerate if either $\chi_1 = \chi_2$ or $\varphi_1 = \varphi_2$, in which case the rectangle (and its diagonals) correspond to the segment itself, or $\chi_1 = \chi_2$ and $\varphi_1 = \varphi_2$, and the rectangle is just a single point.

Example 3.2. Let $\mathbf{i} = (1,2)$, $\mathbf{j} = (3,2)$, $\mathbf{k} = (1,5)$, and $\mathbf{l} = (3,5)$. The tuple is in $\mathcal{D}$ and its corresponding bounding rectangle is a non-degenerate rectangle. The function $M(\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}) = (1,3,2,5)$.

Example 3.3. Let $\mathbf{i} = \mathbf{j} = (1,4)$ and $\mathbf{k} = \mathbf{l} = (3,7)$. The tuple is also in $\mathcal{D}$ and minimal bounding rectangle formed by these points is an interval $\{(1,4),(3,7)\}$. The function $M(\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}) = (1,3,4,7)$.

To start we consider the non-degenerate cases. Fix any $(\chi_1,\chi_2,\varphi_1,\varphi_2)$ with $\chi_1 < \chi_2$ and $\phi_1 < \phi_2$. There are in total $\binom{4}{2}^2 = 36$ tuples $(\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l})$ in $\mathcal{D}$ with $M(\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}) = (\chi_1,\chi_2,\varphi_1,\varphi_2)$. Twenty-four of these tuples correspond to the setting where none of $\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}$ are equal, as there are twenty-four permutations of the assignment of the labels $\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}$ to the four points. (This corresponds to the first example). In this case the four points form a rectangle, and we have $|\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}}| \leq \frac{1}{2}((\vec{v}_{\chi_1,\varphi_1}\vec{v}_{\chi_2,\varphi_2})^2 + (\vec{v}_{\chi_1,\varphi_2}\vec{v}_{\chi_2,\varphi_1})^2)$. Intuitively, in these cases, we assign the "weight" of the tuple to the diagonals.

The remaining twelve tuples in $M^{-1}(\chi_1, \chi_2, \varphi_1, \varphi_2)$ correspond to intervals. (This corresponds to the second example.) In this case two of $\mathbf{i}, \mathbf{j}, \mathbf{k}, \mathbf{l}$ correspond to one endpoint of the interval, and the other two labels correspond to the other endpoint. Hence we have either $|\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}}| = (\vec{v}_{\chi_1,\varphi_1}\vec{v}_{\chi_2,\varphi_2})^2$ or $|\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}}| = (\vec{v}_{\chi_1,\varphi_2}\vec{v}_{\chi_2,\varphi_1})^2$, and there are six tuples for each case.

Therefore for any $\chi_1 < \chi_2 \in [n]$ and $\varphi_1 < \varphi_2 \in [n]$ we have:

$$\sum_{\substack{(\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l})\in \\ M^{-1}(\chi_1,\chi_2,\varphi_1,\varphi_2)}} |v_{\mathbf{i}}v_{\mathbf{j}}v_{\mathbf{k}}v_{\mathbf{l}}| \le 18((v_{\chi_1,\varphi_1}v_{\chi_2,\varphi_2})^2 + (v_{\chi_1,\varphi_2}, v_{\chi_2,\varphi_1})^2).$$

The analysis is similar for the degenerate cases, where the constant 18 in the bound above is now quite loose. When exactly one of $\chi_1 = \chi_2$ or $\varphi_1 = \varphi_2$ holds, the size of $M^{-1}(\chi_1, \chi_2, \varphi_1, \varphi_2)$ is $\binom{4}{2} = 6$, and the resulting intervals correspond to vertical or horizontal lines. When both $\chi_1 = \chi_2$ and $\varphi_1 = \varphi_2$, then $|M^{-1}(\chi_1, \chi_2, \varphi_1, \varphi_2)| = 1$. In sum, we have Following the same analysis as for the non-degenerate cases, we find

$$\begin{aligned}
\sum_{\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}\in\mathcal{D}} |\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}}| &= \sum_{\substack{\chi_1\le\chi_2\\ \varphi_1\le\varphi_2}} \sum_{\substack{(\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l})\in \\ M^{-1}(\chi_1,\chi_2,\varphi_1,\varphi_2)}} |\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}}| \\
\le \sum_{\substack{\chi_1<\chi_2\\ \varphi_1<\varphi_2}} 18((\vec{v}_{\chi_1,\varphi_1}\vec{v}_{\chi_2,\varphi_2})^2 &+ (\vec{v}_{\chi_1,\varphi_2}\vec{v}_{\chi_2,\varphi_1})^2) + \sum_{\substack{\chi_1=\chi_2\\ \varphi_1<\varphi_2}} 6((\vec{v}_{\chi_1,\varphi_1}\vec{v}_{\chi_2,\varphi_2})^2 + (\vec{v}_{\chi_1,\varphi_2}\vec{v}_{\chi_2,\varphi_1})^2) \\
&+ \sum_{\substack{\chi_1<\chi_2\\ \varphi_1=\varphi_2}} 6((\vec{v}_{\chi_1,\varphi_1}\vec{v}_{\chi_2,\varphi_2})^2 + (\vec{v}_{\chi_1,\varphi_2}\vec{v}_{\chi_2,\varphi_1})^2) + \sum_{\substack{\chi_1=\chi_2\\ \varphi_1=\varphi_2}} (\vec{v}_{\chi_1,\varphi_1}\vec{v}_{\chi_2,\varphi_2})^2 \\
&\le 9 \sum_{\substack{\mathbf{i}\in[n]^2\\ \mathbf{j}\in[n]^2}} (\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}})^2 = 9\mathrm{E}^2[Y].
\end{aligned}$$

Finally, we have $\sum_{\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}\in[n]^2} |\mathrm{E}[H(\mathbf{i})H(\mathbf{j})H(\mathbf{k})H(\mathbf{l})]| \cdot |\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}}| \le \sum_{\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}\in\mathcal{D}} |\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}}| \le 9\mathrm{E}^2[Y]$ and $\mathrm{Var}[Y] \le 8\mathrm{E}[Y]^2$. ■

We emphasize the geometric interpretation of the above proof as follows. The goal is to bound the variance by a constant times $\mathrm{E}^2[Y] = \sum_{\mathbf{i},\mathbf{j}\in[n]^2} (\vec{v}_{\mathbf{i}}v_{\mathbf{j}})^2$, where the index set is the set of all possible lines in plane $[n]^2$ (each line appears twice). We first show that $\mathrm{Var}[Y] \le \sum_{\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}\in\mathcal{D}} |\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}}|$, where the 4-tuple index set corresponds to a set of rectangles in a natural way. The main idea of [12] is to use inequalities of the form $|\vec{v}_{\mathbf{i}}\vec{v}_{\mathbf{j}}\vec{v}_{\mathbf{k}}\vec{v}_{\mathbf{l}}| \le \frac{1}{2}((\vec{v}_{\chi_1,\varphi_1}\vec{v}_{\chi_2,\varphi_2})^2 + (\vec{v}_{\chi_1,\varphi_2}\vec{v}_{\chi_2,\varphi_1})^2)$ to assign the "weight" of each 4-tuple to the diagonals of the corresponding rectangle. The above analysis shows that 18 copies of all lines are sufficient to accommodate all 4-tuples. While similar inequalities could also assign the weight of a 4-tuple to the vertical or horizontal edges of the corresponding rectangle, using vertical or horizontal edges is problematic. The reason is that there are $\Omega(n^4)$ 4-tuples but only $O(n^3)$ vertical or horizontal edges, so some lines would receive $\Omega(n)$ weight, requiring $\Omega(n)$ copies. This problem is already noted in [7].

Our bound here is $\mathrm{E}[Y^2] \le 9\mathrm{E}^2[Y]$, while in [12] the bound obtained is $\mathrm{E}[Y^2] \le 3\mathrm{E}^2[Y]$. There appears to have been an error in the derivation in [12]; some intuition comes from the following example. We note that $|\mathcal{D}|$ is at least $\binom{4}{2}^2 \cdot \binom{n}{2}^2 = 9n^4 - 9n^2$. (This counts the number of non-degenerate 4-tuples.) Now if we set $v_i = 1$ for all $1 \le i \le n^2$, we have $\mathrm{E}[Y^2] \ge |\mathcal{D}| = 9n^4 - 9n^2 \sim 9\mathrm{E}^2(D)$, which suggests $\mathrm{Var}[D] > 3\mathrm{E}^2[D]$. Again, we emphasize this discrepancy is of little importance to [12]; the point there is that the variance is bounded by a constant factor times the square of the expectation. It is here, where we are generalizing to $k \geq 3$, that the exact constant factor is of some importance.

Given the bounds on the expectation and variance for the $D_{i,j}$, standard techniques yield a bound on the performance of our algorithm.

Theorem 3.4. *For every $\epsilon > 0$ and $\delta > 0$, there exists a randomized algorithm that computes, given a sequence $(a_1^1, a_1^2), \ldots, (a_m^1, a_m^2)$, in one pass and using $O(\epsilon^{-2} \log \frac{1}{\delta}(\log m + \log n))$ memory bits, a number Med so that the probability Med deviates from $\|v\|^2$ by more than ϵ is at most δ.*

Proof. Recall the algorithm described in the beginning of Section 3: let $s_1 = 72\epsilon^{-2}$ and $s_2 = 2 \log \delta$. We first computes s_2 random variables $Y_1, Y_2, \ldots, Y_{s_2}$ and outputs their median Med, where each Y_i is the average of s_1 random variables Y_{ij}: $1 \leq j \leq s_1$ and Y_{ij} are independent, identically distributed random variables computed by Figure 1. By Chebyshev's inequality, we know that for any fixed i,

$$\Pr\left(\left|Y_i - \|\vec{v}\|\right|\right) \geq \epsilon\|\vec{v}\|] \leq \frac{\mathrm{Var}(Y_i)}{\epsilon^2\|\vec{v}\|^2} = \frac{(1/s_1)\mathrm{Var}[Y]}{\epsilon^2\|\vec{v}\|^2} = \frac{(9\epsilon^2/72)\|\vec{v}\|^2}{\epsilon^2\|\vec{v}\|^2} = \frac{1}{8}.$$

Finally, by standard Chernoff bound arguments (see for example Chapter 4 of [14]), the probability that more than $s_2/2$ of the variables Y_i deviate by more than $\epsilon\|\vec{v}\|$ from $\|\vec{v}\|$ is at most δ. In case this does not happen, the median Med supplies a good estimate to the required quantity $\|\vec{v}\|$ as needed. ■

4. The General Case $k \geq 3$

Now let us move to the general case where $k \geq 3$. Recall that $\vec{v}$ is a vector in R^{n^k} that maintains certain statistics of a data stream, and we are interested in estimating its ℓ_2 norm $\|\vec{v}\|$. There is a natural generalization for Indyk and McGregor's method for $k = 2$ to construct an estimator for $\|\vec{v}\|$: let $h_1, \ldots, h_k : [n] \to \{-1, 1\}$ be independent copies of 4-wise independent hash functions (namely, $h_i(1), \ldots, h_i(n) \in \{-1, 1\}$ are 4-wise independent hash functions for each $i \in [k]$, and $h_1(\cdot), \ldots, h_k(\cdot)$ are mutually independent.). Let $H(\mathbf{p}) = \prod_{i=1}^k h_j(p_j)$. The estimator Y is defined as $Y \equiv \left(\sum_{\mathbf{p}\in[n]^k} \vec{v}_{\mathbf{p}} H(\mathbf{p})\right)^2$.

Our goal is to show that $\mathrm{E}[Y] = \|\vec{v}\|^2$ and $\mathrm{Var}[Y]$ is reasonably small so that a streaming algorithm maintaining multiple independent instances of estimator Y will be able to output an approximately correct estimation of $\|\vec{v}\|$ with high probability. Notice that when $\|\vec{v}\|$ represents the ℓ_2 distance between the joint distribution and the tensors of the marginal distributions, the estimator can be computed efficiently in a streaming model similarly to as in Figure 1. We stress that our result is applicable to a broader class of ℓ_2-norm estimation problems, as long as the vector $\vec{v}$ to be estimated has a corresponding efficiently computable estimator Y in an appropriate streaming model. Formally, we shall prove the following main lemma in the next subsection.

Lemma 4.1. *Let $\vec{v}$ be a vector in R^{n^k}, and $h_1, \ldots, h_k : [n] \to \{-1, 1\}$ be independent copies of 4-wise independent hash functions. Define $H(\mathbf{p}) = \prod_{i=1}^k h_j(p_j)$, and $Y \equiv \left(\sum_{\mathbf{p}\in[n]^k} \vec{v}_{\mathbf{p}} H(\mathbf{p})\right)^2$. We have $\mathrm{E}[Y] = ||\vec{v}||$ and $\mathrm{Var}[Y] \leq 3^k \mathrm{E}[Y]^2$.*

We remark that the bound on the variance in the above lemma is tight. One can verify that when the vector $\vec{v}$ is a uniform vector (i.e., all entries of $\vec{v}$ are the same), the variance of Y is $\Omega(3^k E[Y]^2)$. With the above lemma, the following main theorem mentioned in the introduction immediately follows by a standard argument presented in the proof of Theorem 3.4 in the previous section.

Theorem 4.2. *Let $\vec{v}$ be a vector in $\mathrm{R}^{[n]^k}$ that maintains an arbitrary statistics in a data stream of size m, in which every item is from $[n]^k$. Let $\epsilon, \delta \in (0,1)$ be real numbers. If there exists an algorithm that maintains an instance of Y using $O(\mu(n,m,k,\epsilon,\delta))$ memory bits, then there exists an algorithm Λ such that:*

(1) With probability $\geq 1-\delta$ the algorithm Λ outputs a value between $[(1-\epsilon)\|\vec{v}\|^2, (1+\epsilon)\|\vec{v}|^2]$ and

(2) the space complexity of Λ is $O(3^k \frac{1}{\epsilon^2} \log \frac{1}{\delta} \mu(n,m,k,\epsilon,\delta))$.

As discussed above, an immediate corollary is the existence of a one-pass space efficient streaming algorithm to detect the dependency of k random variables in ℓ_2-norm:

Corollary 4.3. *For every $\epsilon > 0$ and $\delta > 0$, there exists a randomized algorithm that computes, given a sequence $a_1, \ldots, a_m$ of k-tuples, in one pass and using $O(3^k \epsilon^{-2} \log \frac{1}{\delta}(\log m + \log n))$ memory bits, a number Y so that the probability Y deviates from the square of the ℓ_2 distance between product and joint distribution by more than a factor of $(1+\epsilon)$ is at most δ.*

4.1. Analysis of the Sketch Y

This section is devoted to prove Lemma 4.1, where the main challenge is to bound the variance of Y. The geometric approach of Indyk and McGregor [12] presented in Section 3 for the case of $k = 2$ can be extended to analyze the general case. However, we remark that the generalization requires new ideas. In particular, instead of performing "local analysis" that maps each rectangle to its diagonals, a more complex "global analysis" is needed in higher dimensions to achieve the desired bounds. The alternative proof we present here utilizes similar ideas, but relies on a more combinatorial rather than geometric approach.

For the expectation of Y, we have

$$\begin{aligned} \mathrm{E}[Y] &= \mathrm{E}\left[\sum_{\mathbf{p},\mathbf{q}\in[n]^k} \vec{v}_{\mathbf{p}} \cdot \vec{v}_{\mathbf{q}} \cdot H(\mathbf{p}) \cdot H(\mathbf{q})\right] \\ &= \sum_{\mathbf{p}\in[n]^k} \vec{v}_{\mathbf{p}}^2 \cdot \mathrm{E}\left[H(\mathbf{p})^2\right] + \sum_{\mathbf{p}\neq\mathbf{q}\in[n]^k} \vec{v}_{\mathbf{p}} \cdot \vec{v}_{\mathbf{q}} \cdot \mathrm{E}\left[H(\mathbf{p})H(\mathbf{q})\right] \\ &= \sum_{\mathbf{p}\in[n]^k} \vec{v}_{\mathbf{p}}^2 = ||\vec{v}||^2, \end{aligned}$$

where the last equality follows by $H(\mathbf{p})^2 = 1$, and $\mathrm{E}\left[H(\mathbf{p})H(\mathbf{q})\right] = 0$ for $\mathbf{p} \neq \mathbf{q}$.

Now, let us start to prove $\mathrm{Var}[Y] \leq 3^k \mathrm{E}[Y]^2$. By definition, $\mathrm{Var}[Y] = \mathrm{E}[(Y - \mathrm{E}[Y])^2]$, so we need to understand the following random variable:

$$Err \equiv Y - \mathrm{E}[Y] = \sum_{\mathbf{p}\neq\mathbf{q}\in[n]^k} H(\mathbf{p})H(\mathbf{q})\vec{v}_{\mathbf{p}}\vec{v}_{\mathbf{q}}. \tag{4.1}$$

The random variable Err is a sum of terms indexed by pairs $(\mathbf{p},\mathbf{q}) \in [n]^k \times [n]^k$ with $\mathbf{p} \neq \mathbf{q}$. At a very high level, our analysis consists of two steps. In the first step, we group the terms in Err properly and simplify the summation in each group. In the second step, we expand the square of the sum in $\mathrm{Var}[Y] = \mathrm{E}[Err^2]$ according to the groups and apply Cauchy-Schwartz inequality three times to bound the variance.

We shall now gradually introduce the necessary notation for grouping the terms in Err and simplifying the summation. We remind the reader that vectors over the reals (e.g., $\vec{v} \in R^{n^k}$) are denoted by $\vec{v}, \vec{w}, \vec{r}$, and vectors over $[n]$ are denoted by $\mathbf{p}, \mathbf{q}, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ and referred as *index vectors*. We use $S \subseteq [k]$ to denote a subset of $[k]$, and let $\bar{S} = [k] \backslash S$. We use $\mathrm{Ham}(\mathbf{p}, \mathbf{q})$ to denote the *Hamming distance* of index vectors $\mathbf{p}, \mathbf{q} \in [n]^k$, i.e., the number of coordinates where $\mathbf{p}$ and $\mathbf{q}$ are different.

Definition 4.4. *(Projection and inverse projection)* Let $\mathbf{c} \in [n]^k$ be an index vector and $S \subseteq [k]$ a subset. We define the *projection of* $\mathbf{c}$ *to* S, denoted by $\Phi_S(\mathbf{c}) \in [n]^{|S|}$, to be the vector $\mathbf{c}$ restricted to the coordinates in S. Also, let $\mathbf{a} \in [n]^{|S|}$ and $\mathbf{b} \in [n]^{k-|S|}$ be index vectors. We define the *inverse projection of* $\mathbf{a}$ *and* $\mathbf{b}$ *with respect to* S, denoted by $\Phi_S^{-1}(\mathbf{a}, \mathbf{b}) \in [n]^k$, as the index vector $\mathbf{c} \in [n]^k$ such that $\Phi_S(\mathbf{c}) = \mathbf{a}$ and $\Phi_{\bar{S}}(\mathbf{c}) = \mathbf{b}$.

We next define *pair groups* and use the definition to group the terms in Err.

Definition 4.5. *(Pair Group)* Let $S \subseteq [k]$ be a subset of size $|S| = t$. Let $\mathbf{c}, \mathbf{d} \in [n]^t$ be a pair of index vectors with $\mathrm{Ham}(\mathbf{c}, \mathbf{d}) = t$ (i.e., all coordinates of $\mathbf{c}$ and $\mathbf{d}$ are distinct.). The *pair group* $\sigma_S(\mathbf{c}, \mathbf{d})$ is the set of pairs $(\mathbf{p}, \mathbf{q}) \in [n]^k \times [n]^k$ such that (i) on coordinate S, $\Phi_S(\mathbf{p}) = \mathbf{c}$ and $\Phi_S(\mathbf{q}) = \mathbf{d}$, and (ii) on coordinate $\bar{S}$, $\mathbf{p}$ and $\mathbf{q}$ are the same, i.e., $\Phi_{\bar{S}}(\mathbf{p}) = \Phi_{\bar{S}}(\mathbf{q})$. Namely,

$$\sigma_S(\mathbf{c}, \mathbf{d}) = \left\{ (\mathbf{p}, \mathbf{q}) \in [n]^k \times [n]^k : \Big(\mathbf{c} = \Phi_S(\mathbf{p})\Big) \wedge \Big(\mathbf{d} = \Phi_S(\mathbf{q})\Big) \wedge \Big(\Phi_{\bar{S}}(\mathbf{p}) = \Phi_{\bar{S}}(\mathbf{q})\Big) \right\}. \tag{4.2}$$

To give some intuition for the above definitions, we note that for every $\mathbf{a} \in [n]^{|\bar{S}|}$, there is a unique pair $(\mathbf{p}, \mathbf{q}) \in \sigma_S(\mathbf{c}, \mathbf{d})$ with $\mathbf{a} = \Phi_{\bar{S}}(\mathbf{p}) = \Phi_{\bar{S}}(\mathbf{q})$, and so $|\sigma_S(\mathbf{c}, \mathbf{d})| = n^{|\bar{S}|}$. On the other hand, for every pair $(\mathbf{p}, \mathbf{q}) \in [n]^k \times [n]^k$ with $\mathbf{p} \neq \mathbf{q}$, there is a unique non-emtpy $S \subseteq [k]$ such that $\mathbf{p}$ and $\mathbf{q}$ are distinct on exactly coordinates in S. Therefore, $(\mathbf{p}, \mathbf{q})$ belongs to exactly one pair group $\sigma_S(\mathbf{c}, \mathbf{d})$. It follows that we can partition the summation in Err according to the pair groups:

$$Err = \sum_{\substack{S \subseteq [k] \\ S \neq \emptyset}} \sum_{\substack{\mathbf{c}, \mathbf{d} \in [n]^{|S|}, \\ \mathrm{Ham}(\mathbf{c}, \mathbf{d}) = |S|}} \sum_{\substack{(\mathbf{p}, \mathbf{q}) \in \\ \sigma_S(\mathbf{c}, \mathbf{d})}} H(\mathbf{p}) H(\mathbf{q}) \vec{v}_{\mathbf{p}} \vec{v}_{\mathbf{q}}. \tag{4.3}$$

We next observe that for any pair $(\mathbf{p}, \mathbf{q}) \in \sigma_S(\mathbf{c}, \mathbf{d})$, since $\mathbf{p}$ and $\mathbf{q}$ agree on coordinates in $\bar{S}$, the value of the product $H(\mathbf{p})H(\mathbf{q})$ depends only on S, $\mathbf{c}$ and $\mathbf{d}$. More precisely,

$$H(\mathbf{p})H(\mathbf{q}) = \prod_{i \in [k]} h_i(p_i) h_i(q_i) = \left(\prod_{i \in S} h_i(p_i) h_i(q_i) \right) \cdot \left(\prod_{i \in \bar{S}} h_i(p_i)^2 \right) = \prod_{i \in S} h_i(p_i) h_i(q_i),$$

which depends only on S, $\mathbf{c}$ and $\mathbf{d}$ since $\Phi_S(\mathbf{p}) = \mathbf{c}$ and $\Phi_S(\mathbf{q}) = \mathbf{d}$. This motivates the definition of *projected hashing*.

Definition 4.6. *(Projected hashing)* Let $S = \{s_1, s_2, \dots, s_t\}$ be a subset of $[k]$, where $s_1 < s_2 < \cdots < s_j$. Let $\mathbf{c} \in [n]^t$. We define the *projected hashing* $H_S(\mathbf{c}) = \prod_{i \le t} h_{s_i}(c_i)$.

We can now translate the random variable Err as follows:

$$Err = \sum_{\substack{S \subseteq [k] \\ S \neq \emptyset}} \sum_{\substack{\mathbf{c}, \mathbf{d} \in [n]^{|S|}, \\ \mathrm{Ham}(\mathbf{c}, \mathbf{d}) = |S|}} \left(H_S(\mathbf{c}) H_S(\mathbf{d}) \sum_{\substack{(\mathbf{p}, \mathbf{q}) \in \\ \sigma_S(\mathbf{c}, \mathbf{d})}} \vec{v}_{\mathbf{p}} \vec{v}_{\mathbf{q}} \right). \tag{4.4}$$

Fix a pair group $\sigma_S(\mathbf{c}, \mathbf{d})$, we next consider the sum $\sum_{(\mathbf{p}, \mathbf{q}) \in \sigma_S(\mathbf{c}, \mathbf{d})} \vec{v}_{\mathbf{p}} \vec{v}_{\mathbf{q}}$. Recall that for every $\mathbf{a} \in [n]^{|\bar{S}|}$, there is a unique pair $(\mathbf{p}, \mathbf{q}) \in \sigma_S(\mathbf{c}, \mathbf{d})$ with $\mathbf{a} = \Phi_{\bar{S}}(\mathbf{p}) = \Phi_{\bar{S}}(\mathbf{q})$. The sum can be viewed as the inner product of two vectors of dimension $n^{|\bar{S}|}$ with entries indexed by $\mathbf{a} \in [n]^{|\bar{S}|}$. To formalize this observation, we introduce the definition of *hyper-projection* as follows.

Definition 4.7. *(Hyper-projection)* Let $\vec{v} \in R^{n^k}$, $S \subseteq [k]$, and $\mathbf{c} \in [n]^{|S|}$. The *hyper-projection* $\Upsilon_{S,\mathbf{c}}(\vec{v})$ of $\vec{v}$ (with respect to S and $\mathbf{c}$) is a vector $\vec{w} = \Upsilon_{S,\mathbf{c}}(\vec{v})$ in $\mathrm{R}^{[n]^{k-|S|}}$ such that $\vec{w}_{\mathbf{d}} = \vec{v}_{\Phi_S^{-1}(\mathbf{c},\mathbf{d})}$ for all $\mathbf{d} \in [n]^{k-|S|}$.

Using the above definition, we continue to rewrite the Err as

$$Err = \sum_{\substack{S\subseteq[k]\\ S\neq\emptyset}} \sum_{\substack{\mathbf{c},\mathbf{d}\in[n]^{|S|},\\ \mathrm{Ham}(\mathbf{c},\mathbf{d})=|S|}} H_S(\mathbf{c})H_S(\mathbf{d}) \cdot \langle \Upsilon_{S,\mathbf{c}}(\vec{v}), \Upsilon_{S,\mathbf{d}}(\vec{v})\rangle. \tag{4.5}$$

Finally, we consider the product $H_S(\mathbf{c})H_S(\mathbf{d})$ again and introduce the following definition to further simplify the Err.

Definition 4.8. *(Similarity and dominance)* Let t be a positive integer.

- Two pairs of index vectors $(\mathbf{c}, \mathbf{d}) \in [n]^t \times [n]^t$ and $(\mathbf{a}, \mathbf{b}) \in [n]^t \times [n]^t$ are *similar* if for all $i \in [t]$, the two sets $\{c_i, d_i\}$ and $\{a_i, b_i\}$ are equal. We denote this as $(\mathbf{a}, \mathbf{b}) \sim (\mathbf{c}, \mathbf{d})$.
- Let $\mathbf{c}$ and $\mathbf{d} \in [n]^t$ be two index vectors. We say $\mathbf{c}$ *is dominated by* $\mathbf{d}$ if $c_i < d_i$ for all $i \in [t]$. We denote this as $\mathbf{c} \prec \mathbf{d}$. Note that $\mathbf{c} \prec \mathbf{d} \Rightarrow \mathrm{Ham}(\mathbf{c}, \mathbf{d}) = t$.

Now, note that if $(\mathbf{a}, \mathbf{b}) \sim (\mathbf{c}, \mathbf{d})$, then $H_S(\mathbf{a})H_S(\mathbf{b}) = H_S(\mathbf{c})H_S(\mathbf{d})$ since the value of the product $H_S(\mathbf{c})H_S(\mathbf{d})$ depends on the values $\{c_i, d_i\}$ only as a set. It is also not hard to see that $\sim$ is an equivalence relation, and for every equivalent class $[(\mathbf{a}, \mathbf{b})]$, there is a unique $(\mathbf{c}, \mathbf{d}) \in [(\mathbf{a}, \mathbf{b})]$ with $\mathbf{c} \prec \mathbf{d}$. Therefore, we can further rewrite the Err as

$$Err = \sum_{\substack{S\subseteq[k]\\ S\neq\emptyset}} \sum_{\mathbf{c}\prec\mathbf{d}\in[n]^{|S|}} H_S(\mathbf{c})H_S(\mathbf{d}) \cdot \left(\sum_{(\mathbf{a},\mathbf{b})\sim(\mathbf{c},\mathbf{d})} \langle \Upsilon_{S,\mathbf{a}}(\vec{v}), \Upsilon_{S,\mathbf{b}}(\vec{v})\rangle \right). \tag{4.6}$$

We are ready to bound the term $\mathrm{E}[Err^2]$ by expanding the square of the sum according to Equation (4.6). We first show in Lemma 4.9 below that all the cross terms in the following expansion vanish.

$$\mathrm{Var}[Y] = \sum_{\substack{S,S'\subseteq[k]\\ S,S'\neq\emptyset}} \sum_{\substack{\mathbf{c}\prec\mathbf{d}\in[n]^{|S|}\\ \mathbf{c}'\prec\mathbf{d}'\in[n]^{|S|'}}} \mathrm{E}[H_S(\mathbf{c})H_S(\mathbf{d})H_{S'}(\mathbf{c}')H_{S'}(\mathbf{d}')]\cdot$$
$$\left[\left(\sum_{(\mathbf{a},\mathbf{b})\sim(\mathbf{c},\mathbf{d})} \langle \Upsilon_{S,\mathbf{a}}(\vec{v}), \Upsilon_{S,\mathbf{b}}(\vec{v})\rangle \right) \left(\sum_{(\mathbf{a}',\mathbf{b}')\sim(\mathbf{c}',\mathbf{d}')} \langle \Upsilon_{S',\mathbf{a}'}(\vec{v}), \Upsilon_{S',\mathbf{b}'}(\vec{v})\rangle \right) \right]. \tag{4.7}$$

Lemma 4.9. *Let S and S' be subsets of $[k]$, and $\mathbf{c} \prec \mathbf{d} \in [n]^{|S|}$ and $\mathbf{c}' \prec \mathbf{d}' \in [n]^{|S'|}$ index vectors. We have $E[H_S(\mathbf{c})H_S(\mathbf{d})H_{S'}(\mathbf{c}')H_{S'}(\mathbf{d}')] \in \{0, 1\}$. Furthermore, we have* $\mathrm{E}[H_S(\mathbf{c})H_S(\mathbf{d})H_{S'}(\mathbf{c}')H_{S'}(\mathbf{d}')] = 1$ *iff* $(S = S') \wedge (\mathbf{c} = \mathbf{c}') \wedge (\mathbf{d} = \mathbf{d}')$.

Proof. Recall that $h_1, \ldots, h_k$ are independent copies of 4-wise independent uniform random variables over $\{-1, 1\}$. Namely, for every $i \in [k]$, $h_i(1), \ldots, h_i(n)$ are 4-wise independent, and $h_1(\cdot), \ldots, h_k(\cdot)$ are mutually independent. Observe that for every $i \in [k]$, there are at most 4 terms out of $h_i(1), \ldots, h_i(n)$ appearing in the product $H_S(\mathbf{c})H_S(\mathbf{d})H_{S'}(\mathbf{c}')H_{S'}(\mathbf{d}')$. It follows that all distinct terms appearing in $H_S(\mathbf{c})H_S(\mathbf{d})H_{S'}(\mathbf{c}')H_{S'}(\mathbf{d}')$ are mutually independent uniform random variable over $\{-1,1\}$. Therefore, the expectation is either 0, if there is some $h_i(j)$ that appears an odd number of times, or 1, if all $h_i(j)$ appear an even number of times. By inspection, the latter case happens if and only if $(S=S')\wedge(\mathbf{c}=\mathbf{c}')\wedge(\mathbf{d}=\mathbf{d}')$. ■

By the above lemma, Equation (4.7) is simplified to

$$\mathrm{Var}[Y]=\sum_{\substack{S\subseteq[k]\\ S\neq\emptyset}}\sum_{\mathbf{c}\prec\mathbf{d}\in[n]^{|S|}}\left(\sum_{(\mathbf{a},\mathbf{b})\sim(\mathbf{c},\mathbf{d})}\langle\Upsilon_{S,\mathbf{a}}(\vec{v}),\Upsilon_{S,\mathbf{b}}(\vec{v})\rangle\right)^2. \tag{4.8}$$

We next apply the Cauchy-Schwartz inequality three times to bound the above formula. Consider a subset $S\subseteq[k]$ and a pair $\mathbf{c}\prec\mathbf{d}\in[n]^{|S|}$. Note that there are precisely $2^{|S|}$ pairs $(\mathbf{a},\mathbf{b})$ such that $(\mathbf{a},\mathbf{b})\sim(\mathbf{c},\mathbf{d})$. Thus, by the Cauchy-Schwartz inequality:

$$\begin{aligned}\left(\sum_{\substack{(\mathbf{a},\mathbf{b})\in[n]^{|S|}\\(\mathbf{a},\mathbf{b})\sim(\mathbf{c},\mathbf{d})}}\langle\Upsilon_{S,\mathbf{a}}(\vec{v}),\Upsilon_{S,\mathbf{b}}(\vec{v})\rangle\right)^2 &\leq 2^{|S|}\sum_{\substack{(\mathbf{a},\mathbf{b})\in[n]^{|S|}\\(\mathbf{a},\mathbf{b})\sim(\mathbf{c},\mathbf{d})}}(\langle\Upsilon_{S,\mathbf{a}},\Upsilon_{S,\mathbf{b}}\rangle)^2\\ &\leq 2^{|S|}\sum_{\substack{(\mathbf{a},\mathbf{b})\in[n]^{|S|}\\(\mathbf{a},\mathbf{b})\sim(\mathbf{c},\mathbf{d})}}\langle\Upsilon_{S,\mathbf{a}}(\vec{v}),\Upsilon_{S,\mathbf{a}}(\vec{v})\rangle\cdot\langle\Upsilon_{S,\mathbf{b}},\Upsilon_{S,\mathbf{b}}(\vec{v})\rangle.\end{aligned}$$

Notice that in the second inequality, we applied Cauchy-Schwartz in a component-wise manner. Next, for a subset $S\subseteq[k]$, we can apply the Cauchy-Schwartz inequality a third time (from the third line to the fourth line) as follows:

$$\begin{aligned}&\sum_{\mathbf{c}\prec\mathbf{d}\in[n]^{|S|}}\left(\sum_{\substack{(\mathbf{a},\mathbf{b})\in[n]^{|S|}\\(\mathbf{a},\mathbf{b})\sim(\mathbf{c},\mathbf{d})}}\langle\Upsilon_{S,\mathbf{a}}(\vec{v}),\Upsilon_{S,\mathbf{b}}(\vec{v})\rangle\right)^2\\ \leq\ & 2^{|S|}\sum_{\mathbf{c}\prec\mathbf{d}\in[n]^{|S|}}\sum_{\substack{(\mathbf{a},\mathbf{b})\in[n]^{|S|}\\(\mathbf{a},\mathbf{b})\sim(\mathbf{c},\mathbf{d})}}\langle\Upsilon_{S,\mathbf{a}}(\vec{v}),\Upsilon_{S,\mathbf{a}}(\vec{v})\rangle\cdot\langle\Upsilon_{S,\mathbf{b}}(\vec{v}),\Upsilon_{S,\mathbf{b}}(\vec{v})\rangle\\ =\ & 2^{|S|}\sum_{\substack{\mathbf{c},\mathbf{d}\in[n]^{|S|}\\ \mathrm{Ham}(\mathbf{c},\mathbf{d})=|S|}}\langle\Upsilon_{S,\mathbf{c}}(\vec{v}),\Upsilon_{S,\mathbf{c}}(\vec{v})\rangle\cdot\langle\Upsilon_{S,\mathbf{d}}(\vec{v}),\Upsilon_{S,\mathbf{d}}(\vec{v})\rangle\\ \leq\ & 2^{|S|}\sum_{\mathbf{c},\mathbf{d}\in[n]^{|S|}}\langle\Upsilon_{S,\mathbf{c}}(\vec{v}),\Upsilon_{S,\mathbf{c}}(\vec{v})\rangle\cdot\langle\Upsilon_{S,\mathbf{d}}(\vec{v}),\Upsilon_{S,\mathbf{d}}(\vec{v})\rangle\\ =\ & 2^{|S|}\left(\sum_{\mathbf{c}\in[n]^{|S|}}\langle\Upsilon_{S,\mathbf{c}}(\vec{v}),\Upsilon_{S,\mathbf{c}}(\vec{v})\rangle\right)^2.\end{aligned}$$

Finally, we note that by definition, we have $\sum_{\mathbf{c}\in[n]^{|S|}}\langle\Upsilon_{S,\mathbf{c}}(\vec{v}),\Upsilon_{S,\mathbf{c}}(\vec{v})\rangle=||\vec{v}||^2$, which equals to $\mathrm{E}[Y]$. It follows that the variance in Equation (4.8) can be bounded by

$$\mathrm{Var}[Y]\leq\sum_{S\subseteq[k],S\neq\emptyset}2^{|S|}\cdot\mathrm{E}[Y]^2=\mathrm{E}[Y]^2\sum_{i=1}^{k}\binom{k}{i}2^i=(3^k-1)\mathrm{E}[Y]^2,$$

which finishes the proof of Lemma 4.1.

5. Conclusion

There remain several open questions left in this space. Lower bounds, particularly bounds that depend non-trivially on the dimension k, would be useful. There may still be room for better algorithms for testing k-wise independence in this manner using the ℓ_2 norm. A natural generalization would be to find a particularly efficient algorithm for testing k-out-of-s-wise independence (other than handling each set of k variable separately). More generally, a question given in [12], to identify random variables whose correlation exceeds some threshold according to some measure, remains widely open.

References

[1] Noga Alon, Alexandr Andoni, Tali Kaufman, Kevin Matulef, Ronitt Rubinfeld, and Ning Xie. Testing k-wise and almost k-wise independence. In *STOC '07: Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, pages 496–505, New York, NY, USA, 2007. ACM.

[2] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.

[3] Noga Alon, Oded Goldreich, and Yishay Mansour. Almost k-wise independence versus k-wise independence. *Inf. Process. Lett.*, 88(3):107–110, 2003.

[4] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[5] T. Batu, L. Fortnow, E. Fischer, R. Kumar, R. Rubinfeld, and P. White. Testing random variables for independence and identity. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 442, Washington, DC, USA, 2001. IEEE Computer Society.

[6] Tugkan Batu, Ravi Kumar, and Ronitt Rubinfeld. Sublinear algorithms for testing monotone and unimodal distributions. In *STOC '04: Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, pages 381–390, New York, NY, USA, 2004. ACM.

[7] Vladimir Braverman and Rafail Ostrovsky. Measuring k-wise independence of streaming data. *CoRR*, abs/0806.4790v1, 2008.

[8] Vladimir Braverman and Rafail Ostrovsky. Measuring independence of datasets. *CoRR*, abs/0903.0034, 2009.

[9] Vladimir Braverman and Rafail Ostrovsky. AMS without **4**-wise independence on product domains. *CoRR*, abs/0806.4790v3, 2009.

[10] Kai-Min Chung, Zhenming Liu, and Michael Mitzenmacher. Testing k-wise independence over streaming data. Unpublished manuscript, available at `http://www.eecs.harvard.edu/~michaelm/postscripts/sketchexttemp.pdf`, 2009.

[11] Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. Technical report, Electronic Colloquium on Computational Complexity, 2000.

[12] Piotr Indyk and Andrew McGregor. Declaring independence via the sketching of sketches. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 737–745, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[13] E. L. Lehmann and Springer. *Testing Statistical Hypotheses (Springer Texts in Statistics)*. Springer, January 1997.

[14] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[15] Ronitt Rubinfeld and Rocco A. Servedio. Testing monotone high-dimensional distributions. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 147–156, New York, NY, USA, 2005. ACM.

[16] Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *SODA '04: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 615–624, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 131-142
www.stacs-conf.org

QUANTUM ALGORITHMS FOR TESTING PROPERTIES OF DISTRIBUTIONS

SERGEY BRAVYI[1] AND ARAM W. HARROW[2,3] AND AVINATAN HASSIDIM[3]

[1] IBM Watson Research Center, Yorktown Heights, NY 10598 (USA).

[2] Department of Mathematics, University of Bristol, Bristol, BS8 1TW, U.K.

[3] Massachusetts Institute of Technology, Cambridge, MA 02139 (USA)

Abstract. Suppose one has access to oracles generating samples from two unknown probability distributions p and q on some N-element set. How many samples does one need to test whether the two distributions are close or far from each other in the L_1-norm? This and related questions have been extensively studied during the last years in the field of property testing. In the present paper we study quantum algorithms for testing properties of distributions. It is shown that the L_1-distance $\|p-q\|_1$ can be estimated with a constant precision using only $O(N^{1/2})$ queries in the quantum settings, whereas classical computers need $\Omega(N^{1-o(1)})$ queries. We also describe quantum algorithms for testing Uniformity and Orthogonality with query complexity $O(N^{1/3})$. The classical query complexity of these problems is known to be $\Omega(N^{1/2})$. A quantum algorithm for testing Uniformity has been recently independently discovered by Chakraborty et al [14].

1. Introduction

1.1. Problem statement and main results

Suppose one has access to a black box generating independent samples from an unknown probability distribution p on some N-element set. If the number of available samples grows linearly with N, one can use the standard Monte Carlo method to simultaneously estimate the probability p_i of every element $i = 1, \ldots, N$ and thus obtain a good approximation to the entire distribution p. On the other hand, many important questions that one usually encounters in statistical analysis can be answered using only a *sublinear* number of samples. For example, deciding whether p is close in the L_1-norm to another distribution q requires approximately $N^{1/2}$ samples if q is known [8] and approximately $N^{2/3}$ samples if q is also specified by a black-box [9]. Another example is estimating the Shannon entropy $H(p) = -\sum_i p_i \log_2 p_i$. It was shown in [7, 21] that distinguishing whether $H(p) \le a$ or $H(p) \ge b$ requires approximately $N^{\frac{a}{b}}$ samples. Other examples include deciding whether p is close to a monotone or a unimodal distribution [10], and deciding whether a pair of distributions

1998 ACM Subject Classification: G.3 Probabilistic algorithms.

Key words and phrases: quantum computing, property testing, sampling.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2450

have disjoint supports [15]. These and other questions fall into the field of *distribution testing* [6, 21] that studies how many samples one needs to decide whether an unknown distribution has a certain property or is far from having this property. The purpose of the present paper is to explore whether quantum computers are capable of solving distribution testing problems more efficiently.

The black-box sampling model adopted in [8, 9, 7, 10, 6, 21] assumes that a tester is presented with a list of samples drawn from an unknown distribution. What does it mean to sample from an unknown distribution in the quantum settings? Let us start by casting the black-box sampling model into a form that admits a quantum generalization. Suppose p is an unknown distribution on an N-element set $[N] \equiv \{1, \ldots, N\}$ and let S be some specified integer. We shall assume that p is represented by an *oracle* $O_p : [S] \to [N]$ such that the probability p_i of any element $i \in [N]$ is proportional to the number of elements in the pre-image of i, that is, the number of inputs $s \in [S]$ such that $O_p(s) = i$. In other words, one can sample from p by querying the oracle O_p on a random input $s \in [S]$ drawn from the uniform distribution[1]. Note that a tester interacting with an oracle can potentially be more powerful due to the possibility of making adaptive queries which could allow him to learn the internal structure of the oracle as opposed to the black-box model. However, the unstructured nature of the problem we consider means that this advantage is restricted to avoiding repeated queries of the same position. This in turn becomes significant only when $\Omega(S)$ queries are made, which is not relevant in our setting where we have assumed that $S \gg N$. We omit the precise formulation of this claim, which is stated as Lemma 6.1 of [13].

The oracle model admits a standard quantum generalization. Specifically, we shall transform the oracle O_p into a reversible form by keeping a copy of the input and writing the output of O_p into an ancillary register. A quantum oracle generating p is a unitary operator whose action on basis vectors coincides with the reversible version of O_p, as we will explain further in Section 2.

The present paper focuses on testing three particular properties of distributions, namely, *Statistical Difference*, *Orthogonality*, and *Uniformity*. The corresponding property testing problems are promise problems so that a tester is required to give a correct answer (with a bounded error probability) only for those instances that satisfy the promise.

Problem 1.1 (Testing Uniformity).
Instance: Integers N, S, precision $\epsilon > 0$. Access to an oracle generating a distribution p on $[N]$.
Promise: Either p is the uniform distribution or the L_1-distance between p and the uniform distribution is at least ϵ.
Decide which one is the case.

Problem 1.2 (Testing Orthogonality).
Instance: Integers N, S, precision $\epsilon > 0$. Access to oracles generating distributions p, q on $[N]$.
Promise: Either p and q are orthogonal (i.e. have disjoint support) or the L_1-distance between p and q is at most $2 - \epsilon$.
Decide which one is the case.

[1] Although in this model probabilities p_i can only take values that are multiples of $1/S$, choosing sufficiently large S allows one to represent any distribution p with an arbitrarily small error.

Problem 1.3 (Testing Statistical Difference).
Instance: Integers N, S, thresholds $0 \le a < b \le 2$. Access to oracles generating distributions p and q on $[N]$.
Promise: Either $\|p-q\|_1 \le a$ or $\|p-q\|_1 \ge b$.
Decide which one is the case.

We assume that the precision ϵ is bounded from below by a fixed constant independent of N, for instance, $\epsilon \ge 1/10$. The same applies to the decision gap $b-a$ for testing Statistical Difference. Given a function $f(N)$ we shall say that a property is testable in $f(N)$ queries if there exists a testing algorithm making at most $f(N)$ queries that gives a correct answer with a sufficiently high probability (say $2/3$) for any distributions p, q satisfying the promise and for any oracles[2] specifying p and q. If a promise is violated, a tester can give an arbitrary answer.

Our main results are the following theorems.

Theorem 1.4. *Statistical Difference is testable on a quantum computer in $O(N^{1/2})$ queries.*

Theorem 1.5. *Uniformity is testable on a quantum computer in $O(N^{1/3})$ queries.*

Theorem 1.6. *Orthogonality is testable on a quantum computer in $O(N^{1/3})$ queries.*

It is known that classically testing Orthogonality and Uniformity requires $\Omega(N^{1/2})$ queries, see Sections 6.1 and 6.2, while Statistical Difference is not testable in $O(N^{\alpha})$ queries for any $\alpha < 1$, see [21]. Therefore quantum computers provide a polynomial speedup for testing Uniformity, Orthogonality, and Statistical Difference in terms of query complexity.

Testing Orthogonality is closely related to the Collision Problem studied in [12]. In Section 6.1 we describe a randomized reduction from the Collision Problem to testing Orthogonality. Using the quantum lower bound for the Collision Problem due to Aaronson and Shi [2] we obtain the following result.

Theorem 1.7. *Testing Orthogonality on a quantum computer requires $\Omega(N^{1/3})$ queries.*

Quite recently Chakraborty, Fischer, Matsliah, and de Wolf [14] independently discovered a quantum Uniformity testing algorithm with query complexity $O(N^{1/3})$ and proved a lower bound $\Omega(N^{1/3})$ for testing Uniformity. These authors also presented a quantum algorithm for testing whether an unknown distribution p coincides with a known distribution q with query complexity $\tilde{O}(N^{1/3})$.

1.2. Discussion and open problems

One motivation for studying distribution testing problems is that testing Orthogonality and Statistical Difference are complete problems for the complexity class SZK (Statistical Zero Knowledge). More precisely, the following problem known as *Statistical Difference* was shown to be SZK-complete by Vadhan [18]:

Input: *description of classical circuits C_p, C_q that implement oracle functions O_p, O_q : $[S] \to [N]$ and a pair of real numbers $0 \le a < b \le 2$ such that $2a \le b^2$.*
Problem: *Decide whether $\|p-q\|_1 \ge b$ (yes-instance) or $\|p-q\|_1 \le a$ (no-instance)* .

The class SZK includes many interesting algebraic and graph theoretic problems such as Discrete Logarithm, Graph Isomorphism, Graph NonIsomorphism, Quadratic Residuosity,

[2] Note that according to this definition a tester needs at most $f(N)$ queries even in the limit $S \to \infty$.

and The Shortest Vector in Lattice, see [3] and references therein. Thus it is natural to ask whether quantum computers provide a universal speedup for problems in SZK similar to the square-root speedup for problems in NP provided by the Grover search algorithm. Assuming that the circuits C_p, C_q have size $poly(\log{(N)})$, one can easily translate the testing algorithm described in Section 3 to a quantum circuit of size $\tilde{O}(\sqrt{N})$ solving Statistical Difference problem for any constants a, b as above. On the other hand, any classical algorithm treating the circuits C_p, C_q as black boxes would need roughly $N^{1-o(1)}$ queries, see [21], thus requiring a circuit of size $\Omega(N^{1-o(1)})$.

Note that the Statistical Difference problem with $b = 2$ is equivalent to testing Orthogonality. It can be solved classically in time $\tilde{O}(N^{1/2})$ using the classical collision finding algorithm. Unfortunately, the circuit complexity of the quantum Orthogonality testing algorithm described in Section 5 may be different from its query complexity since it uses a quantum membership oracle for a randomly generated set. It is an open problem whether Statistical Difference problem with $b = 2$ can be solved by a quantum circuit of size $\tilde{O}(N^{1/3})$, although with a suitably powerful model of quantum RAM, such membership queries can be done in time $\operatorname{poly}\log(N)$. A related question is that of space-time tradeoffs: our algorithms generally require storing $N^{O(1)}$ classical bits and then querying them with quantum algorithms that use poly(log(N) qubits. We suspect that this amount of storage cannot be reduced without increasing the run-time, but do not have a proof of this conjecture. Similar issues of quantum data structures for set membership and conjectured space-time tradeoffs have arisen for the element distinctness problem[5, 16].

It is worth mentioning that all distribution properties studied in this paper are *symmetric*, that is, these properties are invariant under relabeling of elements in the underlying set $\{1, \ldots, N\}$. Testing symmetric properties of distributions is equivalent to testing properties of functions from $[S]$ to $[N]$ that are invariant under any permutations of inputs and outputs of the function. It was recently shown by Aaronson and Ambainis that quantum computers can provide at most polynomial speedup for testing properties of such symmetric functions [1].

More interesting than the mere fact of polynomial speedups provided by Theorems 1.4, 1.5, 1.6 is the way in which our algorithms achieve it. Classically, the results of Ref. [21] provide a simple characterization of an asymptotically optimal testing algorithm for any symmetric property of a distribution (satisfying certain natural continuity conditions). By contrast, our algorithms use a variety of different strategies both to query the oracles and to analyze the results of those queries. These strategies appear not to be special cases of the quantum walk framework which has been responsible for most of the polynomial quantum speedups found to date [20, 19]. A major challenge for future research is to give a quantum version of Ref. [21]'s Canonical Tester algorithm; in other words, we would like to characterize optimal quantum algorithms for testing any symmetric property of a distribution (or a pair of distributions).

Finally, let us remark that the algorithm for estimating statistical difference described in Section 3 can be easily generalized to construct a quantum algorithm for estimating the von Neumann entropy of a black-box distribution with query complexity $\tilde{O}(N^{1/2})$. Using similar ideas one can construct an $\tilde{O}(N^{1/2})$-time algorithm for estimating the fidelity between two black-box distributions (i.e. $\sum_{i=1}^{N} \sqrt{p_i q_i}$).

The rest of the paper is organized as follows. Section 2 introduces necessary notations and basic facts about the quantum counting algorithm by Brassard, Hoyer, Mosca, and

Tapp [11]. The distribution testing algorithms described in the rest of the paper are actually classical probabilistic algorithms using the quantum counting as a subroutine. Theorem 1.4 is proved in Section 3. Theorem 1.5 is proved in Section 4. Theorem 1.6 is proved in Section 5. We discuss lower bounds for the above distribution testing problems in Section 6.

2. Preliminaries

Let $\mathcal{D}_N$ be the set of probability distributions $p = (p_1, \ldots, p_N)$ such that a probability p_i of any element $i \in [N]$ is a rational number. Let us say that an oracle $O : [S] \to [N]$ generates a distribution $p \in \mathcal{D}_N$ iff for all $i \in [N]$ the probability p_i equals the fraction of inputs $s \in [S]$ such that $O(s) = i$,

$$p_i = \frac{1}{S} \#\{s \in [S] \, : \, O(s) = i\}.$$

Note that the identity of elements in the domain of an oracle O is irrelevant, so if O generates p and σ is any permutation on $[S]$ then $O \circ \sigma$ also generates p. By definition, any map $O : [S] \to [N]$ generates some distribution $p \in \mathcal{D}_N$.

For any oracle $O : [S] \to [N]$ we shall define a quantum oracle $\hat{O}$ by transforming O into a reversible form and allowing it to accept coherent superpositions of queries. Specifically, a quantum oracle $\hat{O}$ is a unitary operator acting on a Hilbert space $\mathbb{C}^S \otimes \mathbb{C}^{N+1}$ equipped with a standard basis $\{|s\rangle \otimes |i\rangle\}$, $s \in [S]$, $i \in \{0\} \cup [N]$ such that

$$\hat{O}\, |s\rangle \otimes |0\rangle = |s\rangle \otimes |O(s)\rangle \quad \text{for all } s \in [S]. \tag{2.1}$$

In other words, querying $\hat{O}$ on a basis vector $|s\rangle \otimes |0\rangle$ one gets the output of the classical oracle $O(s)$ in the second register while the first register keeps a copy of s to maintain unitarity. The action of $\hat{O}$ on a subspace in which the second register is orthogonal to the state $|0\rangle$ can be arbitrary. We shall assume that a quantum tester can execute operators $\hat{O}$, $\hat{O}^\dagger$ and the controlled versions of them. Execution of any one of these operators counts as one query.

Another apparently natural quantum model of a probability distribution is the ability to prepare the state $\sum_{i=1}^N \sqrt{p_i}|i\rangle$; i.e. the ability to "$q$-sample" from the distribution p, c.f. Ref. [3]. However, this ability turns out to be far stronger than the oracle model we will use, since it would allow us to solve Problems 1, 2 and 3 with $O(1)$ q-samples of the distributions p and q. This follows from the well-known result that the observable $\text{SWAP} = \sum_{i,j=1}^N |i,j\rangle\langle j,i|$ has expectation value $|\langle p|q\rangle|^2$ when measured on the state $(\sum_{i=1}^N \sqrt{p_i}|i\rangle) \otimes (\sum_{j=1}^N \sqrt{q_j}|j\rangle)$. Moreover, the ability to efficiently classically sample from a distribution p implies the ability to efficiently construct a quantum oracle $\hat{O}$ corresponding to p, but does not generally imply the ability to q-sample from p. Accordingly, in the rest of the paper we will consider probability distributions to be encoded in quantum oracles.

We shall see that all testing problems posed in Section 1 can be reduced (via classical randomized reductions) to the following problem.

Problem 2.1 (Probability Estimation).
Instance: Integers S, N, description of a subset $A \subset [N]$, precision δ, error probability ω, and access to an oracle generating some distribution $p \in \mathcal{D}_N$. Let $p_A = \sum_{i \in A} p_i$ be the total probability of A.

Task: Generate an estimate $\tilde{p}_A$ satisfying

$$\Pr\left[|\tilde{p}_A - p_A| \le \delta\right] \ge 1 - \omega. \tag{2.2}$$

Our main technical tool will be the quantum counting algorithm by Brassard et al. [11]. Specifically, we shall use the following version of Theorem 12 from [11], whose precise form is proved in [13].

Theorem 2.2. *There exists a quantum algorithm* $\mathbf{EstProb}(p, A, M)$ *taking as input a distribution* $p \in \mathcal{D}_N$ *specified by an oracle, a subset* $A \subset [N]$*, and an integer* M*. The algorithm makes exactly* M *queries to the oracle generating* p *and outputs an estimate* $\tilde{p}_A$ *such that*

$$\Pr\left[|\tilde{p}_A - p_A| \le \delta\right] \ge 1 - \omega \tag{2.3}$$

for all $\delta > 0$ *and* $0 \le \omega \le 1/2$ *satisfying*

$$M \ge \frac{c\sqrt{p_A}}{\omega\delta} \quad \text{and} \quad M \ge \frac{c}{\omega\sqrt{\delta}}. \tag{2.4}$$

Here $c = O(1)$ *is some constant. If* $p_A = 0$ *then* $\tilde{p}_A = 0$ *with certainty.*

(In Eq. 2.4, is is possible to replace $1/\omega$ with $\log(1/\omega)$, but we will not need this improvement.)

3. Quantum algorithm for estimating statistical difference

In this section we sketch the proof of Theorem 1.4. Let $p, q \in \mathcal{D}_N$ be unknown distributions specified by oracles. Define an auxiliary distribution $r \in \mathcal{D}_N$ such that $r_i = (p_i + q_i)/2$ for all $i \in [N]$. If we can sample i from both p and q then by choosing randomly between these two options we can also sample i from r. Let $x \in [0, 1]$ be a random variable which takes value

$$x_i = \frac{|p_i - q_i|}{p_i + q_i}$$

with probability r_i. It is evident that

$$\mathbb{E}(x) = \sum_{i \in [N]} r_i x_i = \frac{1}{2} \sum_{i \in [N]} |p_i - q_i| = \frac{1}{2} \|p - q\|_1. \tag{3.1}$$

Thus in order to estimate the distance $\|p - q\|_1$ it suffices to estimate the expectation value $\mathbb{E}(x)$ which can be done using the standard Monte Carlo method. Since we have to estimate $\mathbb{E}(x)$ only with a constant precision, it suffices to generate $O(1)$ samples of x_i. Given a sample of i (which is easy to generate classically) we can estimate x_i by calling the probability estimation algorithm to get estimates of p_i and q_i. Based on this intuition, we propose the following algorithm for estimating the distance $\|p - q\|_1$.

```
EstDist$(p, q, \epsilon, \tau)$
Set $n = 27/\tau\epsilon^2$, $M = c\sqrt{N}/\epsilon^6\tau^4$.
Let $i_1, \ldots, i_n \in [N]$ be a list of $n$ independent samples drawn from $r$.
For $a = 1, \ldots, n$
{
    Let $\tilde{p}_{i_a}$ be an estimate of $p_{i_a}$ obtained using EstProb$(p, \{i_a\}, M)$.
    Let $\tilde{q}_{i_a}$ be an estimate of $q_{i_a}$ obtained using EstProb$(q, \{i_a\}, M)$.
    Let $\tilde{x}_{i_a} = |\tilde{p}_{i_a} - \tilde{q}_{i_a}|/(\tilde{p}_{i_a} + \tilde{q}_{i_a})$ be our estimate of $x_{i_a}$.
}
Output $\tilde{x} = (1/n)\sum_{a=1}^{n} \tilde{x}_{i_a}$.
```

Here $c = O(1)$ is a constant whose precise value will not be important for us.

Lemma 3.1. *The algorithm* **EstDist**(p, q, ϵ, τ) *outputs an estimate* $\tilde{x}$ *satisfying*

$$\Pr\left[|\tilde{x} - \mathbb{E}(x)| < \epsilon\right] \geq 1 - \tau, \tag{3.2}$$

where $\mathbb{E}(x) = (1/2)\|p - q\|_1$.

The proof can be found in Ref. [13] and is omitted from this extended abstract. The rough idea is that we define an element i to be *bad* iff $\max(p_i, q_i) \leq \tau/3nN$. Then the total probability that any element is bad is $\leq \tau/3$. Conditioned on all the elements being good, we can use Theorem 2.2 to show that we can estimate each p_i and q_i up to multiplicative error $1 - o(1)$, and thereby can also get good estimates of x_i.

Theorem 1.4 follows directly from Lemma 3.1 since **EstDist**(p, q, ϵ, τ) makes $O(\sqrt{N})$ queries to the quantum oracles generating p and q.

4. Quantum algorithm for testing Uniformity

In this section we sketch the proof of Theorem 1.5. Let $p \in \mathcal{D}_N$ be an unknown distribution specified by an oracle. We are promised that either p is the uniform distribution, or p is ϵ-*nonuniform*, that is, the L_1-distance between p and the uniform distribution is at least ϵ. The algorithm described below is based on the following simple observation. Choose some integer $M \ll N$ and let $S = (i_1, \ldots, i_M)$ be a list of M independent samples drawn from the distribution p. Define a random variable $p_S = \sum_{a=1}^{M} p_{i_a}$. It coincides with the total probability of all elements in S unless S contains a collision (that is, $i_a = i_b$ for some $a \neq b$). The characteristic property of the uniform distribution is that $p_S = M/N$ with certainty. On the other hand, we shall see that for any ϵ-nonuniform distribution p_S takes values greater than $(1 + \delta)M/N$ for some constant $\delta > 0$ depending on ϵ with a non-negligible probability. This observation suggests the following algorithm for testing uniformity (the constants K and M below will be chosen later).

```
UTest$(p, K, M, \epsilon)$
    • Let $S = (i_1, \ldots, i_M)$ be a list of $M$ independent samples drawn from $p$.
    • Reject unless all elements in $S$ are distinct.
    • Let $p_S = \sum_{a=1}^{M} p_{i_a}$ be the total probability of elements in $S$.
    • Let $\tilde{p}_S$ be an estimate of $p_S$ obtained using EstProb$(p, S, K)$.
    • If $\tilde{p}_S > (1 + \epsilon^2/8)M/N$ then reject. Otherwise accept.
```

This procedure will need to be repeated several times to achieve the desired bound on the error probability, as we will discuss below.

The main technical result needed is the following lemma.

Lemma 4.1. *Let $p \in \mathcal{D}_N$ be an ϵ-nonuniform distribution. Let $S = (i_1, \ldots, i_M)$ be a list of M independent samples drawn from p, where*

$$M = \left(\frac{32N}{\epsilon^4}\right)^{\frac{1}{3}}. \tag{4.1}$$

Let $p_S = \sum_{a=1}^M p_{i_a}$ and $\alpha = 2^8 \epsilon^{-4}$. Then

$$\Pr\left[p_S \geq (1 + \epsilon^2/2)\frac{M}{N}\right] \geq \frac{1}{2}\exp(-\alpha). \tag{4.2}$$

Theorem 1.4 follows straightforwardly from the above lemma and Theorem 2.2.

Proof of Theorem 1.4. Let M be chosen as in Eq. (4.1) and

$$K = c\frac{e^\alpha N^{1/3}}{\epsilon^{4/3}},$$

where $c = O(1)$ is a constant to be chosen later. Consider the following algorithm:

Perform $L = 4\exp(\alpha)$ independent tests $\mathbf{UTest}(p, K, M, \epsilon)$. If at least one of the tests outputs 'reject' then reject. Otherwise accept.

In the full version of this paper [13], we prove that this algorithm rejects any ϵ-nonuniform distribution with probability at least 2/3 and accepts the uniform distribution with probability at least 2/3.

■

In the rest of this section we sketch the proof of Lemma 4.1 again deferring full proofs to [13]. We shall adopt notations introduced in the statement of Lemma 4.1, that is, the number of samples M is defined by

$$M^3 = 32\epsilon^{-4}N,$$

$\alpha \equiv 2^8\epsilon^{-4}$, $S = (i_1, \ldots, i_M)$ is a list of M independent samples drawn from p, and $p_S = \sum_{a=1}^M p_{i_a}$.

Definition 4.2. An element $i \in [N]$ is called big iff $p_i > 1/(2M^2)$.

Define the set Big $\subset [N]$ of all big elements and their total probability:

$$\mathrm{Big} = \{i \in [N] : p_i > 1/(2M^2)\}, \quad w_{\mathrm{big}} = \sum_{i \in \mathrm{Big}} p_i. \tag{4.3}$$

Also, observe that

$$\mathbb{E}(p_S) = M\langle p|p\rangle \quad \text{and} \tag{4.4a}$$

$$\mathrm{Var}(p_S) = M\left(\sum_{i=1}^N p_i^3 - \langle p|p\rangle^2\right). \tag{4.4b}$$

The proof of Lemma 4.1 is divided into three cases. We shall start by proving the Lemma in the special case when $p \in \mathcal{D}_N$ is ϵ-nonuniform and has no big elements. Using

(4.4), we find that the ϵ-nonuniformity of p implies that $\mathbb{E}(p_S) \geq \frac{M}{N}(1+\epsilon^2)$ while the lack of big elements implies that $\mathrm{Var}(p_S) \leq \langle p|p\rangle/2M$. Then we use Chebyshev's inequality to argue that p_S is likely to be larger than $\frac{M}{N}(1+\epsilon^2/2)$. The second case is when the total weight of big elements is $\leq \alpha/M$, for $\alpha \equiv \epsilon^{-4}/256$. In this case, our sampling is unlikely to encounter any big elements and we can reduce the proof to the case when there are no big elements. Finally, if the total weight of big elements is $> \alpha/M$, then there is a substantial probability that we sample $> \alpha/2$ of them, which will result in p_S being larger than $2M/N$.

5. Quantum algorithm for testing orthogonality

Consider distributions $p, q \in \mathcal{D}_N$ and let $S = (i_1, \ldots, i_M)$ be a list of M independent samples drawn from p. Let $A \subseteq [N]$ be the set of all elements that appear in S at least once. Define the *collision probability*

$$q_A = \sum_{i \in A} q_i.$$

Note that q_A is a deterministic function of A, so the probability distribution of q_A is determined by the probability distribution of A (which depends on p and M). For a fixed A the variable q_A is the probability that a sample drawn from q belongs to A.

Clearly if p and q are orthogonal then $q_A = 0$ with probability 1. On the other hand, if p and q have a constant overlap, we will show that q_A takes values of order M/N with constant probability. Specifically, we shall prove the following lemma.

Lemma 5.1. *Consider a pair of distributions $p, q \in \mathcal{D}_N$ such that $\|p-q\|_1 \leq 2-\epsilon$. Let q_A be a collision probability constructed using M samples. Suppose $M \geq 2^9\epsilon^{-2}$. Then*

$$\Pr\left[q_A \geq \frac{\epsilon^3 M}{2^{11} N}\right] \geq \frac{1}{2}. \tag{5.1}$$

This Lemma suggests the following algorithm for testing orthogonality.

```
OTest$(p, q, M, K)$
    • Let $S = \{i_1, \ldots, i_M\}$ be a list of $M$ independent samples drawn from $p$.
    • Let $A \subseteq [N]$ be the set of elements that appear in $S$ at least once.
    • Let $q_A = \sum_{i \in A} q_i$ be the total probability of elements in $A$ with respect to $q$.
    • Let $\tilde{q}_A$ be estimate of $q_A$ obtained using EstProb$(q, A, K)$.
    • If $\tilde{q}_A \geq \frac{\epsilon^3 M}{2^{12} N}$ then reject. Otherwise accept.
```

We note that if $q_A = 0$ then $\tilde{q}_A = 0$ with certainty (see Theorem 2.2) and so **OTest** accepts any pair of orthogonal distributions with certainty. Again the full proof of Theorem 1.6 is left to [13]. The idea it to choose $M = K = O\left(\frac{N^{1/3}}{\epsilon}\right)$ and apply **OTest**(p, q, M, K) to distributions $p, q \in \mathcal{D}_N$. According to Lemma 5.1, if $\|p-q\|_1 \leq 2-\epsilon$ then $q_A \geq \epsilon^3 M/(2^{11}N)$ with probability $\geq 1/2$. When this holds, the algorithm rejects whenever $|\tilde{q}_A - q_A| \leq \frac{q_A}{2}$ since this implies $\tilde{q}_A \geq q_A/2 \geq \epsilon^3 M/(2^{12}N)$. By Theorem 2.2, our choice of K is sufficient to achieve this with $\Omega(1)$ probability.

It remains only to prove Lemma 5.1.

Proof. Begin by defining two sets of indices:

$$B \equiv \{i : q_i < \frac{\epsilon}{4} p_i\} \qquad \text{and} \qquad C \equiv \{i : p_i \leq \frac{\epsilon}{32} N^{-1}\} \tag{5.2}$$

Let B^c, C^c denote the complements of B and C respectively. We will prove that

$$\Pr\left[|A \cap B^c \cap C^c| \geq \frac{\epsilon}{16} M\right] \geq 1/2, \tag{5.3}$$

which will imply the Lemma since

$$q_A \geq \sum_{i \in A \cap B^c \cap C^c} q_i \geq \frac{\epsilon}{4} \sum_{i \in A \cap B^c \cap C^c} p_i \geq \frac{\epsilon^2}{2^7 N} |A \cap B^c \cap C^c|.$$

This is achieved by using a Chernoff-Hoeffding bound to show that $|A \cap B|$ and $|A \cap C$ are each unlikely to be much larger than their expectations. The details are in [13]. ■

6. Lower bounds

6.1. Reduction from the Collision Problem to testing Orthogonality

One can get lower bounds on the query complexity of testing Orthogonality using the lower bounds for the Collision problem [2]. Indeed, let $H : [N] \to [N]$ be an oracle function such that either H is one-to-one (yes-instance) or H is two-to-one (no-instance). The Collision Problem is to decide which one is the case. It was shown by Refs. [2, 4, 17] that the quantum query complexity of the Collision problem is $\Omega(N^{1/3})$. Below we show that the Collision problem can be reduced to testing Orthogonality. As a result, testing Orthogonality will be shown to require $\Omega(N^{1/2})$ queries classically and $\Omega(N^{1/3})$ queries quantumly.

Indeed, choose a random permutation $\sigma : [N] \to [N]$ and define functions $O_p, O_q : [N/2] \to [3N/2]$ by restricting the composition $H \circ \sigma$ to the subsets of odd and even integers respectively:

$$O_p(s) = H(\sigma(2s-1)), \quad O_q(s) = H(\sigma(2s))$$

where $s \in [N/2]$.

For any yes-instance (i.e. H is one-to-one), the distributions $p, q \in \mathcal{D}_{3N/2}$ generated by O_p and O_q are uniform distributions on some pair of disjoint subsets of $[3N/2]$; that is, p and q are orthogonal.

We need to show that for any no-instance (H is two-to-one) the distance $\|p - q\|_1$ takes values smaller than $2 - \epsilon$ with a sufficiently high probability for some constant ϵ. This is established by the following Lemma, whose proof can be found in [13].

Lemma 6.1. *Let $H : [N] \to [3N/2]$ be any two-to-one function. Let $\sigma : [N] \to [N]$ be a random permutation drawn from the uniform distribution. Then*

$$\Pr\left[\|p - q\|_1 \leq \frac{7}{4}\right] \geq \frac{1}{2}.$$

6.2. Classical lower bound for testing Uniformity

In this section we prove that classically testing Uniformity requires $\Omega(N^{1/2})$. A proof uses the machinery developed by Valiant in [21]. Valiant's techniques apply to testing *symmetric* properties of distributions, that is, properties that are invariant under relabeling of elements in the domain of a distribution. Clearly, Uniformity is a symmetric property.

We shall need two technical tools from [21], namely, the Positive-Negative Distance lemma and Wishful Thinking theorem (see Theorem 4 and Lemma 3 in [21]). Let us start from introducing some notations. Let $p \in \mathcal{D}_N$ be an unknown distribution and $S = (i_1, \ldots, i_M)$ be a list of M independent samples drawn from p. We shall say that S has a collision of order r iff some element $i \in [N]$ appears in S exactly r times. Let c_r be the total number of collisions of order r, where $r \geq 1$. A sequence of integers $\{c_r\}_{r\geq 1}$ is called a *fingerprint* of S. Define a probability distribution D_p^M on a set of fingerprints as follows: (1) draw k from the Poisson distribution $\mathrm{Poi}(k) = e^{-M} M^k/k!$. (2) Generate a list S of k independent samples drawn from p. (3) Output a fingerprint of S.

An important observation made in [21] is that a fingerprint contains all relevant information about a sample list as far as testing symmetric properties is concerned. Thus without loss of generality, a testing algorithm has to make its decision by looking only on a fingerprint of a sample list. Applying Positive-Negative Distance lemma from [21] to testing Uniformity we get the following result.

Lemma 6.2 ([21])**.** *Let u be the uniform distribution on $[N]$ and $p \in \mathcal{D}_N$ be any distribution such that $\|p-u\|_1 \geq 1$. If for some integer M*

$$\|D_p^M - D_u^M\|_1 < \frac{1}{12} \tag{6.1}$$

then Uniformity is not testable in M samples.

The second technical tool is a usable upper bound on the distance between the distributions of fingerprints. For any integer k define an k-th moment of p as $m_k(p) = \sum_{i=1}^N p_i^k$. Clearly $m_k(u) = N^{1-k}$ which is the smallest possible value of a k-th moment for distributions on $[N]$. Applying Wishful Thinking theorem from [21] to testing Uniformity we get the following result (again proved in [13]).

Lemma 6.3 ([21])**.** *Let $p \in \mathcal{D}_N$ be any distribution such that $\|p\|_\infty \leq \delta/M$ for some $\delta > 0$. Then*

$$\|D_p^M - D_u^M\|_1 \leq 40\delta + 10 \sum_{k\geq 2} M^k \,\frac{m_k(p) - N^{1-k}}{\lfloor k/2 \rfloor! \sqrt{1 + M^k\, m_k(p)}}. \tag{6.2}$$

Corollary 6.4. *Uniformity is not testable classically in $32^{-1}\, N^{1/2}$ queries.*

Acknowledgments

We are grateful to Ronald de Wolf for numerous comments that helped to improve the paper. We would like to thank Sourav Chakraborty for informing us about the results in [14]. S.B. thanks CWI for hospitality while this work was being done and was funded by the DARPA QUEST program under contract no. HR0011-09-C-0047. A.W.H. is grateful to IBM and MIT for their hospitality while this work was being done, and is funded by the U.K. EPRSC grant "QIP IRC" and the QAP project (contract IST-2005-15848). A.H. was supported by an xQIT Keck fellowship.

References

[1] S. Aaronson and A. Ambainis. The need of structure in quantum speedups, 2009. arXiv:0911.0996.

[2] S. Aaronson and Y. Shi. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM*, 51(4):595–605, 2004. arXiv:quant-ph/0112086.

[3] D. Aharonov and A. Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In *Proceedings of the 35th Annual ACM Symposium on Theory of computing (STOC)*, pages 20–29. ACM Press New York, NY, USA, 2003. arXiv:quant-ph/0301023.

[4] A. Ambainis. Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. *Theory of Computing*, 1:37–46, 2005. arXiv:quant-ph/0305179.

[5] A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Comput.*, 37(1):210–239, 2007. arXiv:quant-ph/0311001.

[6] T. Batu. *Testing properties of distributions*. PhD thesis, Cornell University, 2001.

[7] T. Batu, S. Dasgupta, R. Kumar, and R. Rubinfeld. The complexity of approximating the entropy. *SIAM J. Comput.*, 35(1):132–150, 2005.

[8] T. Batu, L. Fortnow, E. Fischer, R. Kumar, R. Rubinfeld, and P. White. Testing random variables for independence and identity. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 442, Washington, DC, USA, 2001. IEEE Computer Society.

[9] T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White. Testing that distributions are close. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 259, Washington, DC, USA, 2000. IEEE Computer Society.

[10] T. Batu, R. Kumar, and R. Rubinfeld. Sublinear algorithms for testing monotone and unimodal distributions. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 381–390, New York, NY, USA, 2004. ACM.

[11] G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. In S. J. Lomonaco, editor, *Quantum Computation & Information*, volume 305 of *Contemporary Mathematics Series Millenium Volume*, pages 53–74. AMS, 2002. arXiv:quant-ph/0005055.

[12] G. Brassard, P. Høyer, and A. Tapp. Quantum algorithm for the collision problem. *ACM SIGACT News*, 28:14–19, 1997. arXiv:quant-ph/9705002.

[13] S. Bravyi, A. Harrow, and A. Hassidim. Quantum algorithms for testing properties of distributions, 2009. arXiv:0907.3920.

[14] S. Chakraborty, E. Fischer, A. Matsliah, , and R. de Wolf. Quantum Queries for Testing Distributions, 2009. unpublished.

[15] O. Goldreich and D. Ron. A sublinear bipartiteness tester for bounded degree graphs. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 289–298, New York, NY, USA, 1998. ACM.

[16] L. Grover and T. Rudolph. How significant are the known collision and element distinctness quantum algorithms? *Quant. Inf. & Comp.*, 4:201–206, 2004. arXiv:quant-ph/0309123.

[17] S. Kutin. A quantum lower bound for the collision problem. *Theory of Computing*, 1:29–36, 2005. arXiv:quant-ph/0304162.

[18] A. Sahai and S. Vadhan. A complete promise problem for statistical zero-knowledge. In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, page 448, Washington, DC, USA, 1997. IEEE Computer Society.

[19] M. Santha. Quantum walk based search algorithms. In *TAMC*, volume 4978 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2008. arXiv:0808.0059.

[20] M. Szegedy. Quantum Speed-Up of Markov-Chain-Based Algorithms. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–41, Washington, DC, USA, 2004. IEEE Computer Society.

[21] P. Valiant. Testing symmetric properties of distributions. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 383–392, New York, NY, USA, 2008. ACM.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 143-154
www.stacs-conf.org

OPTIMAL QUERY COMPLEXITY FOR RECONSTRUCTING HYPERGRAPHS

NADER H. BSHOUTY [1] AND HANNA MAZZAWI [2]

[1] Technion, Israel
E-mail address: bshouty@cs.technion.ac.il

[2] Technion, Israel
E-mail address: hanna@cs.technion.ac.il

Abstract. In this paper we consider the problem of reconstructing a hidden weighted hypergraph of constant rank using additive queries. We prove the following: Let G be a weighted hidden hypergraph of constant rank with n vertices and m hyperedges. For any m there exists a non-adaptive algorithm that finds the edges of the graph and their weights using

$$O\left(\frac{m\log n}{\log m}\right)$$

additive queries. This solves the open problem in [S. Choi, J. H. Kim. Optimal Query Complexity Bounds for Finding Graphs. *STOC*, 749–758, 2008].

When the weights of the hypergraph are integers that are less than $O(poly(n^d/m))$ where d is the rank of the hypergraph (and therefore for unweighted hypergraphs) there exists a non-adaptive algorithm that finds the edges of the graph and their weights using

$$O\left(\frac{m\log\frac{n^d}{m}}{\log m}\right).$$

additive queries.

Using the information theoretic bound the above query complexities are tight.

1. Introduction

In this paper we consider the following problem of reconstructing weighted hypergraphs of constant rank[1] (the maximal size of a hyperedge) using additive queries: Let $G=(V,E,w)$ be a weighted hidden hypergraph where $E\subset 2^V$, $|e|$ is constant for all $e\in E$, $w:E\to\mathbb{R}$, and n is the number of vertices in V. Denote by m the size of E. Suppose that the set of vertices V is known and the set of edges E is unknown. Given a set of vertices $S\subseteq V$, an additive query, $Q_G(S)$, returns the sum of weights in the sub-hypergraph induced by S. That is,

$$Q_G(S)=\sum_{e\in E\cap 2^S} w(e).$$

Our goal is to exactly reconstruct the set of edges using additive queries.

[1] Sometimes called dimension.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2496

	Tight Upper Bound	Adaptive Poly. time	Non-adaptive Poly. time
Loops rank$= 1$			
Unweighted Loops	[13, 17, 14, 6]	[8]	OPEN
Bounded Weighted Loops	[11]	OPEN	OPEN
Unbounded Weighted Loops	[10]	OPEN[†]	OPEN[§]
Graph rank$= 2$			
Unweighted Graph	[11]	[22]	OPEN
Bounded Weighted Graph	[11, 9]	OPEN	OPEN
Unbounded Weighted Graph	[10]	OPEN[†]	OPEN
Hypergraph rank> 2			
Unweighted HyperGraph	Ours	OPEN	OPEN
Unbounded Weighted Hypergraph	Ours	OPEN[†]	OPEN

Figure 1: Results for weighted and un-weighted hypergraphs with optimal query complexity. [†]A non-optimal adaptive query complexity algorithm for Hypergraph can be found in [12]. [§] A non-optimal non-adaptive query complexity algorithms can be found in [20] and the references within it.

One can distinguish between two types of algorithms to solve the problem. Adaptive algorithms are algorithms that take into account outcomes of previous queries while non-adaptive algorithms make all queries in advance, before any answer is known. In this paper, we consider non-adaptive algorithms for the problem. Our concern is the query complexity, that is, the number of queries needed to be asked in order to reconstruct the hypergraph.

The hypergraph reconstructing problem has known a significant progress in the past decade. For unweighted hypergraph of rank d the information theoretic lower bound gives

$$\Omega\left(\frac{m\log\frac{n^d}{m}}{\log m}\right)$$

for the query complexity for any adaptive algorithm for this problem.

Many independent results [13, 17, 14, 6][2] have proved a tight upper bound for hypergraph of rank 1, i.e., loops. A tight upper bound was proved for some subclasses of unweighted hypergraphs of rank two, i,e., graphs (Hamiltonian graphs, matching, stars and cliques etc.) [19, 18, 17, 7], unweighted graphs with $\Omega(dn)$ edges where the degree of each vertex is bounded by d [17], graphs with $\Omega(n^2)$ edges [17] and then the former was extended to d-degenerate unweighted graphs with $\Omega(dn)$ edges [19], i.e., graphs that their edges can be changed to directed edges where the out-degree of each vertex is bounded by d. A recent paper by Choi and Kim, [11], gave a tight upper bound for all unweighted graphs. In this paper we give a tight upper bound for all unweighted hypergraphs of constant rank. Our bound is tight even for weighted hypergraphs with integer weights $|w| = poly(n^d/m)$ where d is the rank of the hypergraph.

For weighted hypergraph of constant rank with unbounded weights the information theoretic lower bound gives

$$\Omega\left(\frac{m\log n}{\log m}\right)$$

[2] In [13] Djackov mentions this bound without a proof.

In [11], Choi and Kim prove a tight upper bound for loops (hypergraph of rank 1). For weighted graphs (hypergraph of rank 2) Choi and Kim, [11], proved the following: If $m > (\log n)^\alpha$ for sufficiently large α, then, there exists a non-adaptive algorithm for reconstructing a weighted graph where the weights are real numbers bounded between n^{-a} and n^b for any positive constants a and b using

$$O\left(\frac{m \log n}{\log m}\right)$$

queries.

In [9], Bshouty and Mazzawi close the gap in m and proved that for any weighted graph where the weights are bounded between n^{-a} and n^b for any positive constants a and b and *any* m there exists a non-adaptive algorithm that reconstructs the hidden graph using

$$O\left(\frac{m \log n}{\log m}\right)$$

queries. Then in [10] they extended the result to any weighted graph with any unbounded weights.

In this paper extend all the above results to any hypergraph of constant rank, i.e., the edges of the graph has constant size. This solves the open problems in [11, 9, 10].

The paper is organized as follows: In Section 2, we present notation, basic tools and some background. In Section 3, we prove the main result.

2. Preliminaries

In this section we present some background, basic tools and notation.

For an integer r let $[r]$ be the set $\{1, 2, \ldots, r\}$. For $S \subset [r]$ we define $x^S \in \{0,1\}^r$ where $x_i^S = 1$ if and only if $i \in S$. The inverse operation is $S^x = \{i \mid x_i = 1\}$. We say that $x_1, \ldots, x_d \in \{0,1\}^n$ are *pairwise disjoint* if for every $i \neq j$, we have $x_i * x_j = \mathbf{0}$ where $*$ is component-wise product of two vectors. For a prime p and integers a and b we write $a =_p b$ for $a = b \mod p$. We will also allow $p = \infty$. In this case a and b can be any real numbers and $a =_\infty b$ will mean $a = b$ as real numbers.

2.1. d-Dimensional Matrices

A *d-dimensional matrix* A of *size* $n_1 \times \cdots \times n_d$ over a field $\mathbb{F}$ is a map $A : \prod_{i=1}^d [n_i] \to \mathbb{F}$. We denote by $\mathbb{F}^{n_1 \times \cdots \times n_d}$ the set of all d-dimensional matrices A of size $n_1 \times \cdots \times n_d$. We write $A_{i_1,\ldots,i_d}$ for $A(i_1, \ldots, i_d)$.

The zero map is denoted by $0^{n_1 \times \cdots \times n_d}$. The matrix $B = (A_{i_1,i_2,\ldots,i_d})_{i_1 \in I_1, i_2 \in I_2, \ldots, i_d \in I_d}$ where $I_j \subseteq [n_j]$, is the $|I_1| \times \cdots \times |I_d|$ matrix where $B_{j_1,\ldots,j_d} = A_{\ell_1,\ldots,\ell_d}$ and ℓ_i is the j_ith smallest number in I_i. When $I_j = [n_j]$ we just write j and when $I_j = \{\ell\}$ we just write $j = \ell$. For example, $(A_{i_1,i_2,\ldots,i_d})_{i_1, i_2 = \ell, i_3 \in I_2, \ldots, i_d \in I_d} = (A_{i_1,i_2,\ldots,i_d})_{i_1 \in [n_1], i_2 \in \{\ell\}, i_3 \in I_2, \ldots, i_d \in I_d}$.

When $n_1 = n_2 = \cdots = n_d = n$ then we denote $\mathbb{F}^{n_1 \times \cdots \times n_d}$ by $\mathbb{F}^{\times_d n}$ and $0^{n_1 \times \cdots \times n_d}$ by $0^{\times_d n}$.

We say that the entry $A_{i_1,i_2,\ldots,i_d}$ is of *dimension* r if $|\{i_1, \ldots, i_d\}| = r$. For d-dimensional matrix A we denote by $wt(A)$ the number of points in $\prod_{i=1}^d [n_i]$ that are mapped to non-zero elements in $\mathbb{F}$. We denote by $wt_r(A)$ the number of points in $\prod_{i=1}^d [n_i]$ of dimension r that are mapped to non-zero elements in $\mathbb{F}$. Therefore, $wt(A) = wt_1(A) + wt_2(A) + \cdots + wt_d(A)$.

We denote by $\mathcal{A}_{d,m}$ the set of d-dimensional matrices $A \in \mathbb{F}^{\times_d n}$ where $wt_d(A) \le m$ and $\mathcal{A}^\star_{d,m}$ the set of d-dimensional matrices $A \in \mathbb{F}^{\times_d n}$ where $1 \le wt_d(A) \le m$.

For d-dimensional matrix A of size $n_1 \times \cdots \times n_d$ and $x_i \in \mathbb{F}^{n_i}$ we define

$$A(x_1, \ldots, x_d) = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} A_{i_1,i_2,\ldots,i_d} x_{1i_1} \cdots x_{di_d}.$$

The vector $v = A(\cdot, x_2, \ldots, x_d)$ is n_1-dimensional vector that its ith entry is

$$\sum_{i_2=1}^{n_2} \cdots \sum_{i_d=1}^{n_d} A_{i,i_2,\ldots,i_d} x_{2i_2} \cdots x_{di_d}.$$

For a set of d-dimensional matrices $\mathcal{B}$, a set $S \subseteq (\{0,1\}^n)^d$ is called a *zero test set* for $\mathcal{B}$ if for every $A \in \mathcal{B}$, $A \ne 0$, there is $x \in S$ such that $A(x) \ne 0$.

A d-dimensional matrix is called *symmetric* if for every $i = (i_1, \ldots, i_d) \in [n]^d$ and any permutation ϕ on $[d]$, we have $A_i = A_{\phi i}$, where $\phi i = (i_{\phi(1)}, \ldots, i_{\phi(d)})$. Notice that for a symmetric d-dimensional matrix $A \in \mathbb{F}^{\times_d n}$, $x_i \in \{0,1\}^n$ and any permutation ϕ on $[d]$, we have $A(x_1, \ldots, x_d) = A(x_{\phi(1)}, \ldots, x_{\phi(d)})$.

We will be interested mainly in the fields $\mathbb{F} = \mathbb{R}$ the field of real numbers and $\mathbb{F} = \mathbb{Z}_p$ the field of integers modulo p and in matrices of constant $d = O(1)$ dimension. Also $p > d!$. Although it seems that we are restricting the parameters, the final result has no restriction on the parameters except for $d = O(1)$. We will also abuse the notations $\mathbb{Z}_p$ and $=_p$ and allow $p = \infty$ (so in this paper ∞ is also prime number). In that case $\mathbb{Z}_\infty = \mathbb{R}$ and $=_\infty$ is equality in the filed of real numbers.

2.2. Hypergraph

A *hypergraph* G is a pair $G = (V, E)$ where $V = [n]$ is a set of elements, called nodes or vertices, and E is a set of non-empty subsets of 2^V called *hyperedges* or *edges*. The *rank* $r(G)$ of a hypergraph G is the maximum cardinality of any of the edges in the hypergraph. A hypergraph is called *d-uniform* if all of its edges are of size d.

A *weighted hypergraph* $G = (V, E, w)$ over $\mathbb{Z}_p$ is a hypergraph (V, E) with a weight function $w : E \to \mathbb{Z}_p$. For two weighted hypergraph $G_1 = (V, E_1, w_1)$ and $G_2 = (V, E_2, w_2)$ we define the weighted hypergraph $G_1 - G_2 = (V, E, w)$ where $E = \{e \in E_1 \cup E_2 \mid w_1(e) \ne w_2(e)\}$, and for every $e \in E$, $w(e) = w_1(e) - w_2(e)$. Obviously, $G_1 = G_2$ if and only if $G_1 - G_2$ is an independent set, i.e., $E = \emptyset$.

We denote by $\mathcal{G}_d$ the set of all weighted hypergraphs over $\mathbb{Z}_p$ of rank at most d, $\mathcal{G}_{d,m}$ the set of all weighted hypergraphs over $\mathbb{Z}_p$ of rank at most d and at most m edges and $\mathcal{G}^\star_{d,m}$ the set of all weighted hypergraphs over $\mathbb{Z}_p$ of rank d and at most m edges.

Let $w^\star : 2^V \to \mathbb{Z}_p$ be w extended to all possible edges where for $e \in E$, $w^\star(e) = w(e)$ and for $e \notin E$, $w^\star(e) = 0$.

An *adjacency d-dimensional matrix of a weighted hypergraph G* is a d-dimensional matrix A^G_d where $d \ge r(G)$ such that for every set $e = \{i_1, i_2, \ldots, i_\ell\}$ of size at most d we have $A^G_{d(j_1,\ldots,j_d)} =_p w^\star(e)/N(d,\ell)$ for all $j_1, \ldots, j_d$ such that $\{j_1, j_2, \ldots, j_d\} = \{i_1, \ldots, i_\ell\}$ where

$$N(d,\ell) = \sum_{i=0}^{\ell} (-1)^i \binom{\ell}{i} (\ell - i)^d.$$

That is, $N(d,\ell)$ is the number of possible sequences $(j_1,\ldots,j_d)$ such that $\{j_1,\ldots,j_d\} = \{i_1,\ldots,i_\ell\}$. Note that $N(d,\ell) \le d! < p$ and therefore $N(d,\ell) \ne_p 0$ and A_d^G is well defined.

It is easy to see that the adjacency matrix of a weighted hypergraph is a symmetric matrix and $r(G) = r$ if and only if the adjacency matrix of G has an non-zero entry of dimension r and all entries of dimension greater than r are zero.

2.3. Additive Model

In the Additive Model the goal is to exactly learn a hidden hypergraph with minimal number of additive queries. Given a set of vertices $S \subseteq V$, an *additive query*, $Q_G(S)$, returns the sum of weights in the subgraph induces by S. That is, $Q_G(S) =_p \sum_{e\in E\cap 2^S} w(e)$. Our goal is to exactly reconstruct the set of edges and find their weights using additive queries. See the many applications of this problem in [7, 11, 12].

We say that the set $\mathcal{S} = \{S_1, S_2, \cdots, S_k\} \subseteq 2^V$ is a *detecting set* for $\mathcal{G}_{d,m}$ if for any hypergraph $G \in \mathcal{G}_{d,m}$ there is S_i such that $Q_G(S_i) \ne 0$. We say that the set $\mathcal{S} = \{S_1, S_2, \cdots, S_k\} \subseteq 2^V$ is a *search set* for $\mathcal{G}_{d,m}$ if for any two distinct hypergraphs $G_1, G_2 \in \mathcal{G}_{d,m}$ there is S_i such that $Q_{G_1}(S_i) \ne Q_{G_2}(S_i)$. That is, given $(Q_G(S_i))_i$ one can uniquely determines G. We now prove the following,

Lemma 2.1. *If $\mathcal{S} = \{S_1, S_2, \cdots, S_k\} \subseteq 2^V$ is a detecting set for $\mathcal{G}_{d,2m}$ then it is a search set for $\mathcal{G}_{d,m}$.*

Proof. Let $G_1, G_2 \in \mathcal{G}_{d,m}$ be two distinct weighted hypergraphs. Let $G = G_1 - G_2$. Since $G \in \mathcal{G}_{d,2m}$ there must be $S_i \in \mathcal{S}$ such that $Q_G(S_i) \ne 0$. Since $Q_G(S_i) = Q_{G_1}(S_i) - Q_{G_2}(S_i)$ we have $Q_{G_1}(S_i) \ne Q_{G_2}(S_i)$. ■

2.4. Algebraic View of the Model

It is easy to show that for any hypergraph G of rank r the adjacency d-dimensional matrix of G, A_d^G, for $d \ge r$, is symmetric, contains a nonzero entry of dimension r and

$$Q_G(S) =_p A_d^G(x^S, x^S, \overset{d}{\ldots}, x^S) \overset{\Delta}{=} B_d^G(x^S).$$

For a symmetric d-dimensional matrix A let $B(x) =_p A(x, x, \overset{d}{\ldots}, x)$ where $x \in \{0,1\}^n$. When $x_1, \ldots, x_d \in \{0,1\}^n$ are pairwise disjoint the following lemma shows that $A(x_1,\ldots,x_d)$ can be found by 2^d values of B.

Lemma 2.2. *If $x_1, \ldots, x_d \in \{0,1\}^n$ are pairwise disjoint then*

$$A(x_1,\ldots,x_d) =_p \frac{1}{d!} \sum_{I\in 2^{[d]}} (-1)^{d-|I|} B\left(\sum_{i\in I} x_i\right).$$

Proof. Since

$$A(x_1 + x_1', x_2, \ldots, x_d) =_p A(x_1, x_2, \ldots, x_d) + A(x_1', x_2, \ldots, x_d)$$

and

$$A(x_1, x_2, \ldots, x_d) =_p A(x_{\phi(1)}, x_{\phi(2)}, \ldots, x_{\phi(d)})$$

for any permutation ϕ on $[d]$, the result is analogous to the fact that

$$y_1 y_2 \cdots y_d =_p \frac{1}{d!} \sum_{I \in 2^{[d]}} (-1)^{d-|I|} \left(\sum_{i \in I} y_i \right)^d , \tag{2.1}$$

for formal variables $y_1, \ldots, y_d$. Now notice that

$$\left(\sum_{i \in I} y_i \right)^d =_p \sum_{q_1 + \cdots + q_d = d} \chi\left[\{i | q_i \neq 0\} \subseteq I \right] \binom{d}{q_1\ q_2\ \cdots q_d} y_1^{q_1} \cdots y_d^{q_d},$$

where $\chi[L] = 1$ if the statement L is true and 0 otherwise. Therefore, the coefficient of $y_1^{q_1} \cdots y_d^{q_d}$ in the right hand side of (2.1) is

$$\begin{aligned} & \sum_{I \in 2^{[d]}} (-1)^{d-|I|} \chi\left[\{i | q_i \neq 0\} \subseteq I \right] \binom{d}{q_1\ q_2\ \cdots q_d} \\ =_p & \binom{d}{q_1\ q_2\ \cdots q_d} \sum_{I \in 2^{[d]}} (-1)^{d-|I|} \chi\left[\{i | q_i \neq 0\} \subseteq I \right]. \end{aligned}$$

Now if $\ell = |\{i | q_i \neq 0\}| < d$ then

$$\sum_{I \in 2^{[d]}} (-1)^{d-|I|} \chi\left[\{i | q_i \neq 0\} \subseteq I \right] =_p \sum_{i=\ell}^{d} (-1)^{d-i} \binom{d-\ell}{i-\ell} =_p \sum_{i=0}^{d-\ell} (-1)^{d-\ell-i} \binom{d-\ell}{i} = 0.$$

If $\ell = |\{i | q_i \neq 0\}| = d$ then $q_1 = q_2 = \cdots = q_d = 1$ and

$$\sum_{I \in 2^{[d]}} (-1)^{d-|I|} \chi\left[\{i | q_i \neq 0\} \subseteq I \right] =_p 1.$$

This implies the result. ■

Let G be a hypergraph of rank d and $G^{(i)}$, $i \leq d$, be the sub-hypergraph of G that contains all the edges in G of size i then

Lemma 2.3. *If $x_1, \ldots, x_d \in \{0,1\}^n$ are pairwise disjoint then, we have that $A_d^G(x_1, \ldots, x_d) = A_d^{G^{(d)}}(x_1, \ldots, x_d)$. In particular, if $r(G) < d$ then $A_d^G(x_1, \ldots, x_d) = 0$.*

Proof. Since $x_1, \ldots, x_d \in \{0,1\}^n$ are pairwise disjoint we have

$$\begin{aligned} A_d^G(x_1, \ldots, x_d) &= \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \frac{w^\star(\{i_1, i_2, \ldots, i_d\})}{N(d, |\{i_1, i_2, \ldots, i_d\}|)} x_{1i_1} \cdots x_{di_d} \\ &= \sum_{|\{i_1, \ldots, i_d\}| = d} \frac{w^\star(\{i_1, i_2, \ldots, i_d\})}{N(d,d)} x_{1i_1} \cdots x_{di_d} \\ &= A_d^{G^{(d)}}(x_1, \ldots, x_d). \end{aligned}$$

Now when $r(G) < d$ then $G^{(d)}$ is an independent set (has no edges) and $A_d^{G^{(d)}} = 0$. Then $A_d^G(x) = A_d^{G^{(d)}}(x) = 0$. ■

We now prove

Lemma 2.4. *Let $\Phi_d = \{z_1^{(d)}, \ldots, z_{k_d}^{(d)}\} \subset (\{0,1\}^n)^d$ where for every i the vectors $z_{i,1}^{(d)}, \ldots, z_{i,d}^{(d)}$ are pairwise disjoint. If Φ_d is a zero test set for $\mathcal{A}^\star_{d,(d!)m}$ then*

$$S^{\Phi_d} \triangleq \left\{ S^{y_J} \;\middle|\; y_J = \sum_{j\in J} z_{i,j}^{(d)},\ J \subset [d] \right\}$$

is a detecting set for $\mathcal{G}^\star_{d,m}$.

Proof. Let Φ_d be a zero test set for $\mathcal{A}^\star_{d,(d!)m}$. Let $G \in \mathcal{G}^\star_{d,m}$. Then $A_d^G \neq 0$ and $A_d^G \in \mathcal{A}^\star_{d,(d!)m}$. Therefore, for every $G \in \mathcal{G}^\star_{d,m}$ there is $z_i^{(d)}$ such that $A_d^G(z_i^{(d)}) \neq 0$. By Lemma 2.2,

$$A_d^G(z_i^{(d)}) =_p \frac{1}{d!} \sum_{J\in 2^{[d]}} (-1)^{d-|J|} B_d^G\left(\sum_{j\in J} z_{i,j}^{(d)}\right) \neq 0,$$

and therefore for some $J_0 \subset [d]$,

$$B_d^G\left(\sum_{j\in J_0} z_{i,j}^{(d)}\right) \neq 0,$$

which implies that $Q_G\left(S^{y_{J_0}}\right) \neq 0$ for $y_{J_0} = \sum_{j\in J_0} z_{i,j}^{(d)}$. ∎

We now show

Lemma 2.5. *A detecting set for $\mathcal{G}_{d,m}$ over $\mathbb{Z}_p$ is a detecting set for $\mathcal{G}_{d,m}$ over $\mathbb{R}$.*

Proof. Consider a detecting set $\mathcal{S} = \{S_1, S_2, \cdots, S_k\} \subseteq 2^V$ for $\mathcal{G}_{d,m}$ over $\mathbb{Z}_p$. Consider a $k \times q$ matrix M where

$$q = \sum_{i=0}^{d} \binom{n}{i}$$

that its columns are labelled with sets in $2^{[n]}$ of size at most d and for every $S \subset [n]$ of size at most d we have $M[i,S] = 1$ if $S \subseteq S_i$ and 0 otherwise. Consider for every graph $G \in \mathcal{G}_{d,m}$ a q-vector v_G that its entries are labelled with subsets of $[n]$ of size at most d and $v_G[S] = w^\star(S)$. The labels in v_G are in the same order as the labels of the columns of M. Then it is easy to see that

$$Mv_G =_p (Q_G(S_1), \ldots, Q_G(S_k))^T.$$

Since $Mv_G \neq_p 0$ for every $v_G \in \mathbb{Z}_p^q$ of weight at least one and at most m, every m columns in M are linearly independent over $\mathbb{Z}_p$. Since the entries of M are zeros and ones every m columns in M are linearly independent over $\mathbb{R}$. Therefore,

$$Mv_G = (Q_G(S_1), \ldots, Q_G(S_k))^T \neq 0,$$

for every $v_G \in \mathbb{R}^q$ of weight at least 1 and at most m. ∎

2.5. Distributions

In this subsection we give a distribution that will be used in this paper.

The *uniform disjoint distribution* $\Omega_{d,n}(x)$ over $(\{0,1\}^n)^d$ is defined as

$$\Omega_{d,n}(x) = \left\{ \begin{array}{cl} \frac{1}{(d+1)^n} & x_1, \ldots, x_d \text{ is pairwise disjoint.} \\ 0 & \text{otherwise.} \end{array} \right.$$

In order to choose a random vector x according to the uniform disjoint distribution, one can randomly independently uniformly choose n elements $w_1, w_2, \ldots, w_n$ where $w_i \in [d+1]$ and define the following vector $x = (x_1, x_2, \ldots, x_d) \in (\{0,1\}^n)^d$:

$$x_{ji} = \left\{ \begin{array}{ll} 1 & j = w_i \text{ and } w_i \in [d] \\ 0 & \text{otherwise.} \end{array} \right.$$

We call any index $k \in [n]$ such that $x_{jk} = 0$ for all $j \in [d]$ a *free index*. Let $\Gamma_{d,n} \subset (\{0,1\}^n)^d$ be the set of all pairwise disjoint d-tuple.

2.6. Preliminary Results

In this section we prove,

Lemma 2.6. *Let $A \in \mathbb{F}^{\times_d n} \backslash \{0^{\times_d n}\}$ be an adjacency d-dimensional matrix of a hypergraph G of rank d. Let $x = (x_1, x_2, \ldots, x_d) \in (\{0,1\}^n)^d$ be a randomly chosen d-tuple, that is chosen according to the distribution $\Omega_{d,n}$. Then*

$$\Pr_{x \in \Omega_{d,n}} [A(x) = 0] \leq 1 - \frac{1}{(d+1)^d}.$$

Proof. Let $e = \{i_1, \ldots, i_d\}$ be an edge of size $|e| = d$ and let $x'_j = (x_{j,i_1}, \ldots, x_{j,i_d})$. Consider $\phi(x'_1, \ldots, x'_d)$ that is equal to $A(x)$ with some fixed $x_{j,i} = \xi_{j,i} \in \{0,1\}$ for $i \notin e$. Since $A(x)$ contains the monomial $M = x_{1,i_1} x_{2,i_2} \cdots x_{d,i_d}$ and no other monomial in $A(x)$ contains it, ϕ contains monomial M and therefore $\phi(x'_1, \ldots, x'_d) \not\equiv 0$. If we substitute $x_{j_1, i_{j_2}} = 0$ in ϕ for all $j_1 \neq j_2$ we still get a nonzero function $\phi'(x_{1,i_1}, x_{2,i_2}, \cdots, x_{d,i_d})$ that contains M. Therefore, there is $\xi = (\xi_{1i_1}, \xi_{2i_2}, \cdots, \xi_{di_d}) \in \{0,1\}^d$ such that $\phi'(\xi) \neq 0$. The probability that $(x_{1,i_1}, x_{2,i_2}, \cdots, x_{d,i_d}) = \xi$ and $x_{j_1, i_{j_2}} = 0$ for all $j_1 \neq j_2$ is $(1/d+1)^d$. This implies the result. ■

We will also use the following two lemmas from [9, 10].

Lemma 2.7. *Let $a \in \mathbb{Z}_p^n$ be a non-zero vector, where $p > wt(a)$ is a prime number. Then for a uniformly randomly chosen vector $x \in \{0,1\}^n$ we have*

$$\Pr_x [a^T x =_p 0] \leq \frac{1}{wt(a)^\beta},$$

where $\beta = \frac{1}{2+\log 3} = 0.278943 \cdots$.

Let ι be a function on non-negative integers defined as follows: $\iota(0) = 1$ and $\iota(i) = i$ for $i > 0$.

Lemma 2.8. *Let $m_1, m_2, \ldots, m_t$ be integers in $[m] \cup \{0\}$ such that $m_1 + m_2 + \cdots + m_t = \ell \geq t$. Then $\prod_{i=0}^t \iota(m_i) \geq m^{\lfloor (\ell - t)/(m-1) \rfloor}$.*

3. Reconstructing Hypergraphs

In this section we prove,

Theorem 3.1. *There is a search set for $\mathcal{G}_{d,m}$ over $\mathbb{R}$ of size $k = O\left(\frac{m \log n}{\log m}\right)$.*

Theorem 3.2. *There is a search set for $\mathcal{G}'_{d,m}$ over $\mathbb{R}$ of size $k = O\left(\frac{m \log \frac{n^d}{m}}{\log m}\right)$, where $\mathcal{G}'_{d,m}$ denotes the set of all weighted hypergraphs over $\mathbb{R}$ of rank at most d, at most m edges and weights that are integers bounded by $w = poly(n^d/m)$.*

Proof. We give the proof of Theorem 3.1. The proof of Theorem 3.2 is similar. More details in the full paper.

Let $m < p < 2m$ be a prime number. Suppose there is a zero test set from $\Gamma_{d,n}$ for $\mathcal{A}^\star_{d,m}$ over $\mathbb{Z}_p$ of size $T(n,m,d)$. By Lemma 2.4, there is a detecting set for $\mathcal{G}^\star_{d,m}$ over $\mathbb{Z}_p$ of size $2^d T(n,(d!)m,d)$. Therefore, by Lemma 2.3, there is a detecting set for $\mathcal{G}_{d,m}$ over $\mathbb{Z}_p$ of size $T'(n,m,d) = \sum_{\ell=1}^d 2^\ell T(n,(\ell!)m,\ell)$. By Lemma 2.5, there is a detecting set for $\mathcal{G}_{d,m}$ over $\mathbb{R}$ of size $T'(n,m,d)$. Finally, by Lemma 2.1, there is a search set for $\mathcal{G}_{d,m}$ over $\mathbb{R}$ of size $T'(n,2m,d)$. Now for constant d, if

$$T(n,m,d) = O\left(\frac{m \log n}{\log m}\right), \tag{3.1}$$

then $T'(n,2m,d) = O(T(n,m,d))$. Therefore it is enough to prove the following.

Lemma 3.3. *Let p be a prime number such that $m < p < 2m$. There exists a set $S = \{x_1, x_2, \ldots, x_k\} \subseteq (\{0,1\}^n)^d$ where $x_i = (x_{i,1}, \ldots, x_{i,d}) \in \Gamma_{d,n}$ for $i \in [k]$ and*

$$k = O\left(\frac{m \log n}{\log m}\right),$$

such that: for every d-dimensional matrix $A \in \mathbb{Z}_p^{\times_d n} \setminus \{0^{\times_d n}\}$ with $1 \le wt_d(A) \le m$ there exists an i such that $A(x_i) \ne_p 0$.

Proof. Since $wt_d(A) > 1$ the matrix A has at least one nonzero entry of dimension d. We will assume that all the entries of dimension less than d are zero, that is, $wt(A) = wt_d(A)$. This is because, by Lemma 2.3, the entries of dimension less than d have no effect when the vectors $x_i \in \Gamma_{d,n}$.

We divide the set of such matrices $\mathcal{A} = \{A \mid A \in \mathbb{Z}_p^{\times_d n} \setminus \{0^{\times_d n}\}$ and $wt(A) \le m\}$ into $d+1$ (non-disjoint) sets:

- $\mathcal{A}_0$: The set of all non-zero matrices $A \in \mathbb{Z}_p^{\times_d n}$ such that $wt(A) \le m/\log m$.
- $\mathcal{A}_j$ for $j = 1, \ldots, d$: The set of all non-zero matrices $A \in \mathbb{Z}_p^{\times_d n}$ such that $m \ge wt(A) > m/\log m$ and there are at least
$$\left(\frac{m}{\log m}\right)^{1/d}$$
non-zero elements in $I_j = \{i_j | \exists (i_1, i_2, \ldots, i_{j-1}, i_{j+1}, \ldots, i_d) : A_{i_1,i_2,\ldots,i_d} \ne 0\}$.

Note that $I = \{(i_1, i_2, \ldots, i_d) | A_{i_1,i_2,\ldots,i_d} \ne 0\} \subseteq I_1 \times I_2 \times \cdots \times I_d$ and therefore either $I = wt(A) \le m/\log m$ or there is j such that $|I_j| > (m/\log m)^{1/d}$. Therefore, $\mathcal{A} = \mathcal{A}_0 \cup \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_d$.

Using the probabilistic method, we give $d+1$ sets of pairwise disjoint tuples of vectors $S_0, S_1, \ldots, S_d$ such that for every $j \in \{0\} \cup [d]$ and $A \in \mathcal{A}_j$ there exists a d-tuple x in S_j such that $A(x) \neq 0$ and

$$|S_0| + |S_1| + \cdots + |S_d| = O\left(\frac{m \log n}{\log m}\right).$$

Case 1: $A \in \mathcal{A}_0$: For a random d-tuple x, chosen according to the distribution $\Omega_{d,n}$ we have that

$$\Pr_x[A(x) =_p 0] \leq 1 - \frac{1}{(d+1)^d}.$$

If we randomly choose

$$k_1 = \frac{cm \log n}{\log m}$$

d-tuples, $x_1, \ldots, x_{k_1}$, according to the distribution $\Omega_{d,n}$, then the probability that $A(x_i) = 0$ for all $i \in [k_1]$ is

$$\Pr[\forall i \in [k_1] : A(x_i) =_p 0] \leq \left(1 - \frac{1}{(d+1)^d}\right)^{k_1}.$$

Therefore, by union bound, the probability that there exists a matrix $A \in \mathcal{A}_0$ such that $A(x_i) = 0$ for all $i \in [k_1]$ is

$$\begin{aligned}\Pr[\exists A \in \mathcal{A}_0, \forall i \in [k_1] : A(x_i) =_p 0] &\leq \binom{n^d}{\frac{m}{\log m}} p^{\frac{m}{\log m}} \left(1 - \frac{1}{(d+1)^d}\right)^{\frac{cm \log n}{\log m}} \\ &< n^{d\frac{m}{\log m}} n^{\frac{m}{\log m}} n^{-\frac{c'cm}{\log m}} < 1,\end{aligned}$$

for some constant c. This implies the result.

Case2: $A \in \mathcal{A}_j$ where $j = 1, \ldots, d$: We will assume w.l.o.g that $j = 1$. We first prove the following lemma

Lemma 3.4. *Let $U \subseteq \mathbb{Z}_p^{\times_{d-1} n}$ be the set of all $d-1$-dimensional matrices with weight smaller than $m^{d/(d+1)}$. For $A \in U$ let $\Upsilon(A) \subseteq [n]$ be following set*

$$\Upsilon(A) = \{j \,|\, \exists A_{i_1, i_2, \ldots, i_{d-1}} \neq 0 \text{ and } j \notin \{i_1, i_2, \ldots, i_{d-1}\}\}.$$

Define $Q = \{(A, j) \,|\, A \in U \text{ and } j \in \Upsilon(A)\}$. Then, there is a constant c_0 such that for every $C > c_0$ and

$$k_2 = C \frac{m \log n}{\log m}$$

there exists a multi-set of $d-1$-tuples of (0,1)-vectors $Z = \{z_1, z_2, \ldots, z_{k_2}\} \subseteq (\{0,1\}^n)^{d-1}$ such that for every $(A, j) \in Q$ the size of the set

$$Z_{(A,j)} = \{i \,| A(z_i) \neq 0 \text{ and } j \text{ is a free index}\}$$

is at least $\frac{k_2}{2d^d}$.

Proof. Let $z_i = (z_{i,1}, z_{i,2}, \ldots, z_{i,d-1}) \in (\{0,1\}^n)^{d-1}$ be random $d-1$-tuple of $(0,1)$-vector chosen according to the distribution $\Omega_{d-1,n}$. For $(A,j) \in Q$, and by Lemma 2.6, we have

$$\Pr_{z_i \in \Omega_{d-1,n}}[A(z_i) \neq 0 \text{ and } j \text{ is a free}] = \Pr[j \text{ is free}] \Pr[A(z_i) \neq 0 | j \text{ is free}] \geq \frac{1}{d} \cdot \frac{1}{d^{d-1}} = \frac{1}{d^d}.$$

Therefore, the expected size of $Z_{(A,j)}$ is greater than $\frac{k_2}{d^d}$. By Chernoff bound, if we randomly choose all z_i, $i \in [k_2]$ according to the distribution $\Omega_{d-1,n}$, then, we have

$$\Pr\left[|Z_{(A,j)}| \leq \frac{k_2}{2d^d}\right] \leq e^{\frac{-k_2}{8d^d}}.$$

Thus, the probability that there exists $(A,j) \in Q$ such that $|Z_{(A,j)}| \leq \frac{k_2}{2d^d}$ is

$$\begin{aligned} \Pr\left[\exists (A,j) \in Q : |Z_{(A,j)}| \leq \frac{k_2}{2d^d}\right] &\leq \frac{|Q|}{e^{\frac{-k_2}{8d^d}}} \leq \frac{|U \times [n]|}{e^{\frac{-k_2}{8d^d}}} \leq \frac{n\binom{n^{d-1}}{m^{d/(d+1)}} p^{m^{d/(d+1)}}}{e^{\frac{Cm \log n}{8d^d \log m}}} \\ &\leq \frac{n\binom{n^{d-1}}{m^{d/(d+1)}} n^{m^{d/(d+1)}}}{n^{\frac{C(\log e)m}{8d^d \log m}}} \leq \frac{n^{O\left(m^{d/(d+1)}\right)}}{n^{\frac{Cc'm}{\log m}}} < 1, \end{aligned}$$

for large enough C. This implies the result. ■

Now, Let U and Q be the sets we defined in Lemma 3.4. Let $A \in \mathcal{A}_1$. Since $wt(A) \leq m$ there are at most $m^{1/(d+1)}$ $d-1$-dimensional matrices $(A_{i_1,i_2,\ldots,i_d})_{i_1=j,i_2,\ldots,i_d}$ with weight greater than $m^{d/(d+1)}$. Therefore, there is at least

$$q = \left(\frac{m}{\log m}\right)^{1/d} - m^{1/(d+1)}$$

indices j such that $(A_{i_1,i_2,\ldots,i_d})_{i_1=j,i_2,\ldots,i_d} \in U$. Let U' contain any q indices such that $(A_{i_1,i_2,\ldots,i_d})_{i_1=j,i_2,\ldots,i_d} \in U$. Let A_U be the matrix

$$(A_{i_1,i_2,\ldots,i_d})_{i_1 \in U', i_2,\ldots,i_d}.$$

Let $z_1, z_2, \ldots, z_{k_2} \in (\{0,1\}^n)^{d-1}$ be the set we proved its existence in Lemma 3.4. We now choose $x_i \in \{0,1\}^n$, $i \in [k_2]$ in the following way: Take z_i. For every free index j, choose x_{ij} to be "1" with probability 1/2 and "0" with probability 1/2 (independently for every j). All other entries in x_i are zero, that is, all entries that correspond to non-free index j in z_i are zero. Let $u \in \{0,1\}^n$ be a vector where $u_j = 1$ if $j \in U'$ and zero otherwise. Also, for a $d-1$-tuple z_i let $v_i \in \{0,1\}^n$ be the vector where $v_{ij} = 1$ if j is a free index in z_i and $v_{ij} = 0$ otherwise. By Lemma 2.7 we have that

$$\Pr_x[A(x_i, z_i) =_p 0] \leq \prod_i \frac{1}{\iota(wt(v_i * A(\cdot, z_i)))^\beta} \leq \prod_i \frac{1}{\iota(wt(v_i * (u * A(\cdot, z_i))))^\beta}. \tag{3.2}$$

Note that, A is a hypergraph, thus, for every j such that $(A_{i_1,i_2,\ldots,i_d})_{i_1=j,i_2,\ldots,i_d} \in U$, we have that $((A_{i_1,i_2,\ldots,i_d})_{i_1=j,i_2,\ldots,i_d}, j) \in Q$. Therefore,

$$\sum_i wt(v_i * (u * A(\cdot, z_i))) \geq \frac{qk_2}{2d^d}.$$

Using Lemma 2.8 we have

$$\prod_i \iota(wt(v_i * (u * A(\cdot, z_i)))) \geq q^{\left\lfloor \frac{\frac{qk_2}{2d^d} - k_2}{q-1} \right\rfloor} = m^{c_1 k_2}.$$

Therefore, using (3.2), $\Pr_x[A(x_i,z_i) =_p 0] \le \frac{1}{m^{c_1\beta k_2}}$. Thus, the probability that there exists a matrix $A \in \mathcal{A}_1$ such that for all $i \in [k_2]$ we have $A(x_i,z_i) = 0$ is

$$\Pr_x[A(x_i,z_i) =_p 0] \le \frac{|\mathcal{A}_1|}{m^{c_1\beta k_2}} \le \frac{\binom{n^d}{m}p^m}{m^{c_1\beta k_2}} \le \frac{n^{dm}n^m}{m^{c_1\beta k_2}} < 1,$$

for large enough constant. This implies Lemma 3.3. ∎

This completes the proof of Theorem 3.1. ∎

References

[1] M. Aigner. Combinatorial Search. *John Wiley and Sons*, 1988.

[2] N. Alon and V. Asodi. Learning a Hidden Subgraph. *SIAM J. Discrete Math*, 18, 4, 697–712, 2005.

[3] N. Alon, R. Beigel, S. Kasif, S. Rudich and B. Sudakov. Learning a Hidden Matching. *SIAM J. Comput.* 33, 2, 487–501, 2004.

[4] D. Angluin. and J. Chen. Learning a Hidden Graph Using O($\log n$) Queries per Edge. *COLT*, 210–223, 2004.

[5] D. Angluin and J. Chen. Learning a Hidden Hypergraph. *Journal of Machine Learning Research*, 7, 2215–2236, 2006.

[6] E. Biglieri and L. Györfi. Multiple Access Channels Theory and Practice Volume 10 NATO Security through Science Series - D: Information and Communication Security, April 2007.

[7] M. Bouvel, V. Grebinski, G. Kucherov: Combinatorial Search on Graphs Motivated by Bioinformatics Applications: A Brief Survey. *WG*, 16–27, 2005.

[8] N. H. Bshouty. Optimal Algorithms for the Coin Weighing Problem with a Spring Scale. COLT, 2009.

[9] N. H. Bshouty and H. Mazzawi. Reconstructing Weighted Graphs with Minimal Query Complexity. ALT, 2009.

[10] N. H. Bshouty and H. Mazzawi. On Parity Check $(0,1)$-Matrix over $\mathbb{Z}_p$. TR09-067, ECCC, 2009.

[11] S. Choi, J. H. Kim. Optimal Query Complexity Bounds for Finding Graphs. *STOC*, 749–758, 2008.

[12] S. Choi, K. Jung, J. H. Kim. Almost Tight Upper Bound for Finding Fourier Coefficients of Bounded Pseudo- Boolean Functions. COLT 2008, 123-134, 2008.

[13] A. G. Djackov. On a search model of false coins. In Topics in Information Theory (Colloquia Mathematica Societatis Janos Bolyai 16). Budapest, Hungary: Hungarian Acad. Sci., pp. 163–170, 1975.

[14] A. G. D'yachkov, V. V. Rykov. On a Coding Model for a Multiple-Access Adder Channel, *Probl. Peredachi Inf.*, 17:2 , pp. 2638, 1981.

[15] D. Du and F. K. Hwang. Combinatorial group testing and its application, Volume 3 of Series on applied mathematics. *World Science*, 1993.

[16] P. Erdös. On a lemma of Littlewood and Offord. *Bulletin of the American Mathematical Society*, 51, 898–902, 1945.

[17] V. Grebinski and G. Kucherov. Optimal Reconstruction of Graphs Under the Additive Model. *Algorithmica* , 28(1), 104–124, 2000.

[18] V. Grebiniski and G. Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discrete Applied Mathematics*, 88, 147–165, 1998.

[19] V. Grebinski. On the Power of Additive Combinatorial Search Model. *COCOON*, 194–203 , 1998.

[20] P. Indyk, M. Ruzic. Near-Optimal Sparse Recovery in the L1 Norm. FOCS, 199-207, 2008.

[21] J. E. Littlewood and A. C. Offord. On the number of real roots of a random algebraic equation. III. *Mat. Sbornik*, 12, 277–285, 1943.

[22] H. Mazzawi. Optimally Reconstructing Weighted Graphs Using Queries. SODA, 2010.

[23] L. Reyzin and N. Srivastava. Learning and Verifying Graphs using Queries with a Focus on Edge Counting. *ALT*, 2007.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 155-166
www.stacs-conf.org

ULTIMATE TRACES OF CELLULAR AUTOMATA

JULIEN CERVELLE[1] AND ENRICO FORMENTI[2] AND PIERRE GUILLON[3]

[1] Université Paris-Est, LACL, EA 4219, 61 Av du Général de Gaulle, 94010 Créteil Cedex, France
E-mail address: julien.cervelle@univ-paris-est.fr

[2] Laboratoire I3S, Université de Nice-Sophia Antipolis, 2000, Rte des Lucioles - Les Algorithmes - bât. Euclide B - BP 121, 06903 Sophia Antipolis Cedex, France
E-mail address: enrico.formenti@unice.fr

[3] DIM - CMM, UMI CNRS 2807, Universidad de Chile, Av. Blanco Encalada 2120, 8370459 Santiago, Chile
E-mail address: pguillon@dim.uchile.cl

Abstract. A cellular automaton (CA) is a parallel synchronous computing model, which consists in a juxtaposition of finite automata (cells) whose state evolves according to that of their neighbors. Its trace is the set of infinite words representing the sequence of states taken by some particular cell. In this paper we study the ultimate trace of CA and partial CA (a CA restricted to a particular subshift). The ultimate trace is the trace observed after a long time run of the CA. We give sufficient conditions for a set of infinite words to be the trace of some CA and prove the undecidability of all properties over traces that are stable by ultimate coincidence.

Introduction

Cellular automata are a formal computing model known to display many different dynamical behaviors, from the most simple like nilpotency or equicontinuity to the more complex ones like transitivity, mixing or expansivity. These different behaviors together with their ability to capture many features of natural phenomena increase their popularity in the computer sciencists, mathematicians and physicians communities.

A cellular automaton consists in finite state automata (cells) distributed on a regular lattice (or more generally, on any graph). Each cell updates its state depending on the states of a fixed finite number of neighboring cells. This dependency is given by a local rule which is common to all cells.

In this paper, we resume our study of traces of cellular automata, that is to say the sequence of states taken by one particular cell. The main motivation for this work is to study the way scientists deduce general laws from experiments. They proceed by making

1998 ACM Subject Classification: F.1.1 Models of Computation; F.4.3 Formal Languages.

Key words and phrases: discrete dynamical systems, cellular automata, symbolic dynamics, sofic systems, formal languages, decidability.

Thanks to the Projet Blanc ANR *EMC* and the Comité *ECOS-Sud*.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2451

experimental observations using a finite number of observation variables (*i.e.* a trace in the context of CA). From these observations, they conjecture the mathematical law that rules the whole phenomenon. If this law is verified by (almost all) observations, then the scientist concludes that this is the way the phenomenon behaves, until contradicted by new experiments.

However, one also needs formal results ensuring the correctness of the procedure. Indeed, can any observed trace be generated by a CA? How "large" should a trace be to ensure correct reconstruction of the CA local rule?

The notion of trace for a CA has been studied in [CFG07, CG07]. In this paper, we proceed with two generalizations: partial traces and ultimate traces. A partial trace is the trace of a CA restricted to a particular subshift. This kind of trace is motivated by the fact that there are some experiments where not all initial configurations are admissible: some local constraints have to be respected (e.g. a sand grain cannot be above an empty cell or two positively charged particles cannot be too close to one another *etc.*). The ultimate trace is the trace for the long term behavior *i.e.* when the transient part of the phenomenon is neglected, which is often the case in experimental sciences.

The notion of trace is strictly connected with the concept of symbolic factor. Recall that given a CA $(A^{\mathbb{Z}}, F)$, the system $(B^{\mathbb{N}}, G)$ is a (symbolic) *factor* of $(A^{\mathbb{Z}}, F)$, if there exists a continuous surjection $\varphi : A^{\mathbb{Z}} \to B^{\mathbb{N}}$ such that $\varphi \circ F = G \circ \varphi$. Studying the dynamics of factors is often simpler than studying the original system. Indeed, traces are special cases of factor systems. They were introduced as a form of "back-ingeneering" tool to lift properties of factors to CA. Along this research direction, in Section 5, we prove a Rice's theorem for traces. This is an improvement of a similar result in [CG07], in the sense that it is more "natural" and covers more properties than the previous one.

The paper is organized into three parts. Section 1 recalls main definitions concerning cellular automata and symbolic dynamics. Sections 2 to 4 concern new results about traces. Section 5 presents a Rice-like theorem for traces.

1. Definitions

Let id denote the identity map. If F is a function on a set X, denote $F_{|Y}$ its restriction to some subset $Y \subset X$. If F and G are functions on sets X and Y, then $F \times G$ will denote the function on the cartesian product $X \times Y$ which maps any (x, y) to $(f(x), g(y))$.

Configurations. A *configuration* is a bi-infinite sequence of letters, that is an element of $A^{\mathbb{Z}}$. The set $A^{\mathbb{Z}}$ of configurations is the *phase space*. For integers i, j, denote $[i, j]$ the set $\{i, \dots, j\}$, $[i, j[$ the set $[i, j-1]$, etc... For $x \in A^{\mathbb{Z}}$ and $I = \{i_0, \dots, i_k\} \subset \mathbb{N}$, $i_0 < \dots < i_k$, note $x_I = x_{i_0} \dots x_{i_k}$. Moreover, for a word u, we note $u \sqsubset x$ if u is a factor of x, that is if there exists i and j such that $u = x_{[i,j]}$. If $u \in A^+$, $|u|$ denotes its length, and $x = u^\infty$ [resp. $x = {}^\infty u^\infty$] is the infinite word [resp. configuration] such that $x_{[i,i+|u|[} = u$ for any i in $\mathbb{N}$ [resp. $\mathbb{Z}$]. A word or a configuration is *uniform* if it is made of a single repeated letter. If $L \subset A^k$ and $k \in \mathbb{N} \setminus \{0\}$, we shall also note ${}^\infty L^\infty$ the set of configurations x such that $x_{[ki,(k+1)i[}$ is in L for all $i \in \mathbb{Z}$. Note that we shall assimilate the sets $A^{\mathbb{Z}} \times B^{\mathbb{Z}}$ and $(A \times B)^{\mathbb{Z}}$, for alphabets A, B.

Topology. We endow the phase space with the *Cantor topology.* A base for open sets is given by cylinders: for $j, k \in \mathbb{N}$ and a finite set W of words of length j, we will note $[W]_k$ the *cylinder* $\left\{ w \in A^{\mathbb{Z}} \middle| w_{[k,k+j[} \in W \right\}$. $[W]_k^C$ is the complement of $[W]_k$.

Cellular automata. A (one-dimensional) *cellular automaton* is a parallel synchronous computation model (A, m, d, f) consisting of cells distributed over a regular lattice indexed by $\mathbb{Z}$. Each cell $i \in \mathbb{Z}$ has a state x_i in the finite alphabet A, which evolves depending on the state of their neighbors $x_{[i-m,i-m+d[}$ according to the *local rule* $f : A^d \to A$. The integers $m \in \mathbb{Z}$ and $d > 0$ are the *anchor* and the *diameter* of the CA, respectively. If the anchor is 0, the automaton is said to be *one-sided.* In this case, a cell is only updated according to its state and the ones of its right neighbors. The *global function* of the CA (or simply the CA) is $F : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ such that $F(x)_i = f(x_{[i-m,i-m+d[})$ for every $x \in A^{\mathbb{Z}}$ and $i \in \mathbb{Z}$. The *space-time diagram* of initial configuration $x \in A^{\mathbb{Z}}$ is the sequence of the configurations $(F^j(x))_{j\in\mathbb{N}}$. When the neighborhood of the CA is symmetrical, instead of speaking of anchor and diameter, we shall simply give a *radius.* A CA of radius $r \in \mathbb{N} \setminus \{0\}$, has r for anchor and $2r + 1$ for diameter.

Shifts and subshifts. The *twosided shift* [resp. *onesided shift*], denoted σ, is a particular CA global function defined by $\sigma(x)_i = x_{i+1}$ for every $x \in A^{\mathbb{Z}}$ and $i \in \mathbb{Z}$ [resp. $x \in A^{\mathbb{N}}$ and $i \in \mathbb{N}$] . According to the Hedlund theorem [Hed69], the global functions of CA are exactly the continuous self-maps of $A^{\mathbb{Z}}$ commuting with the twosided shift.

A *twosided subshift* Σ is a closed subset of $A^{\mathbb{Z}}$ with $\sigma(\Sigma) = \Sigma$. A *onesided subshift* Σ is a closed subset of $A^{\mathbb{N}}$ with $\sigma(\Sigma) \subset \Sigma$. We simply speak about the *shift* or *subshifts* when the context allows to understand if it is twosided or onesided.

The language of Σ is $\mathcal{L}(\Sigma) = \{ w \in A^* | \exists z \in \Sigma, w \sqsubset z \}$ and characterizes Σ, since $\Sigma = \left\{ z \in A^{\mathbb{N}} \middle| \forall w \sqsubset z, w \in \mathcal{L}(\Sigma) \right\}$. For $k \in \mathbb{N}$, denote $\mathcal{L}_k(\Sigma) = \mathcal{L}(\Sigma) \cap A^k$.

A subshift Σ is *sofic* if $\mathcal{L}(\Sigma)$ is a regular language, or equivalently if Σ is the set of labels of infinite paths in some edge-labeled graph. In this case, such a graph is called a *graph of* Σ.

A subshift is characterized by its language $\mathcal{F} \subset A^*$ of *forbidden words*, *i.e.* such that $\Sigma = \left\{ z \in A^{\mathbb{N}} \middle| \forall u \in \mathcal{F}, u \not\sqsubset z \right\}$. A subshift is of *finite type* (SFT for short) if its language of forbidden words is finite. It is a k-SFT (for $k \in \mathbb{N}$) if it has a set of forbidden words of length k. For $\Sigma \subset A^{\mathbb{Z}}$, define $\mathcal{O}_\sigma(\Sigma) = \bigcup_{i\in\mathbb{Z}} \sigma^i(\Sigma)$.

Partial cellular automata. A *partial CA* is the restriction of some CA to some twosided subshift.

Subshift projections. If $B \subset A^k$ is an alphabet and $0 \le q < k$, then the q^{th} *projection* of an infinite word $x \in B^{\mathbb{N}}$ is noted $\pi_q(x) \in A^{\mathbb{N}}$ and defined by $\pi_q(x)_j = a_q$ when $x_j = (a_0, \ldots, a_{k-1})$. If Σ is a subshift on B, we also note $\pi(\Sigma) = \bigcup_{0\le q<k} \pi_q(\Sigma)$, which is a subshift on A.

2. Tracebility

Definition 2.1 (Traceability)**.** A subshift $\Sigma \subset A^{\mathbb{N}}$ is *traceable* if there exists a CA F on alphabet A whose *trace* $\tau_F = \left\{ (F^j(x)_0)_{j\in\mathbb{N}} \middle| x \in A^{\mathbb{Z}} \right\}$ is Σ. In this case, we say that F *traces* Σ. If F can be computed effectively from data D, we say that Σ is traceable *effectively from* D. In this notion, D can be any mathematical objet, possibly infinite, provided it has a finite representation (SFT, sofic subshifts, regular languages, CA). In this case, it means one of these representations.

Deterministic subshifts. Given some $\xi : A \to A$, we call *deterministic subshift* the subshift $\mathcal{O}_\xi = \left\{ (\xi^j(a))_{j\in\mathbb{N}} \in A^{\mathbb{Z}} \middle| a \in A \right\}$. The following proposition comes from an easy remark on the evolution of uniform configurations – see Example 4.2 for a subshift which is not traceable.

Proposition 2.2 ([CFG07])**.** *Any traceable subshift $\Sigma \subset A^{\mathbb{N}}$ contains a deterministic subshift $\mathcal{O}_\xi$ for some $\xi : A \to A$.*

Nilpotent subshifts. A subshift $\Sigma \subset A^{\mathbb{N}}$ is 0-*nilpotent* (or simply *nilpotent*) if $0 \in A$ and there is some $j \in \mathbb{N}$ such that $\sigma^j(\Sigma)$ is the singleton $\{0^\infty\}$. It is *weakly nilpotent* if there is some state $0 \in A$ such that for every infinite word $z \in \Sigma$, there is some $j \in \mathbb{N}$ such that $\sigma^j(z) = 0^\infty$. Note that a sofic subshift is weakly nilpotent if and only if it admits a unique periodic infinite word, which is uniform.

The following gives another necessary condition for being the trace of a CA.

Theorem 2.3 ([GR08])**.** *A traceable subshift cannot be weakly nilpotent without being nilpotent.*

Polytraceability. When performing some "back-engineering" from a trace over an alphabet A, *i.e.* when trying to deduce from the trace which CA could have produced it, it is sometimes easier to design a CA over an alphabeth $B \subseteq A^k$ (for some integer k). Being stacked one atop the other, letters of B can be seen as columns of letters of A. In the constructions, the first column is used to produce *all* the elements of Σ and the other columns are used to store elements that help to simulate all possible paths along some graph of Σ. This idea leads to the following notion.

Definition 2.4 (Polytraceabilty)**.** A subshift $\Sigma \subset A^{\mathbb{N}}$ is *polytraceable* if there exists a CA F of anchor 0 and diameter 2 on alphabet $B \subset A^k$ for some k whose *polytrace* $\overset{\circ}{\tau}_F = \bigcup_{0\leq i<k} \pi_i(\tau_F)$ is Σ. In this case, we say that F *polytraces* Σ. If, furthermore, $B = A^k$, we say that the subshift is *totally polytraceable.* If F and B can be computed effectively from data D, we say that Σ is (totally) polytraceable *effectively from D.*

Note that a polytrace cannot be weakly nilpotent without being nilpotent, otherwise it would alors be the case of the corresponding trace. On the other hand, it need not contain a deterministic subshift.

Theorem 2.5 ([CFG07])**.** *Any subshift Σ which is either of finite type or sofic uncountable is polytraceable effectively from Σ.*

CDD subshifts. A sufficient condition for traceability can be given with the help of the following definition. A subshift $\Sigma \subset A^{\mathbb{N}}$ has *cycle distinct from deterministic* property (CDD) if it contains some deterministic subshift $\mathcal{O}_\xi$ and some periodic infinite word w^∞ such that w contains one letter not in $\xi(A)$. We say that Σ is a CDD subshift.

Lemma 2.6 ([CFG07])**.** *Let $\xi : A \to A$ and $\Sigma \subset A^{\mathbb{N}}$ a polytraceable subshift containing a periodic word w^∞, with $w \in A^+ \setminus \xi(A)^+$. Then $\Sigma \cup \mathcal{O}_\xi$ is traceable effectively from ξ, w and a CA polytracing Σ.*

This lemma, together with Theorem 2.5, gives the following result.

Theorem 2.7 ([CFG07])**.** *Any CDD subshift which is either of finite type or sofic uncountable is traceable effectively from the subshift.*

3. Partial traceability

We already discussed about partial traceability in the introduction. Here is the formal definition.

Definition 3.1 (Partial traceability)**.** A subshift Σ is *partially traceable* if there exists a partial CA F on an SFT Γ whose *trace* $\tau_F = \left\{ (F^j(x)_0)_{j\in\mathbb{N}} \middle| x \in \Gamma \right\}$ is Σ. In this case, we say that F *partially traces* (or simply *traces*) Σ. If F and some graph of Γ can be computed effectively from data D, we say that Σ is partially traceable *effectively from* D.

Assume that Σ is polytraced by some CA $G : B^{\mathbb{Z}} \to B^{\mathbb{Z}}$, with $B \subset A^h$ and $h \in \mathbb{N} \setminus \{0\}$ – for instance obtained from Theorem 2.5. We simulate it by a partial CA F on some SFT Λ in order to get a partial trace instead of a polytrace. This is a kind of *ungrouping* operation that splits *macrocells* (on B) into independent cells (on A).

Ungrouping. The *ungrouping* operation represents a standard encoding of configurations of $B^{\mathbb{Z}}$, with $B \subset A^h$ and $h \in \mathbb{N} \setminus \{0\}$, into configurations of $A^{\mathbb{Z}}$ and it is defined as follows

$$\boxplus_h : \begin{array}{rcl} B^{\mathbb{Z}} & \to & A^{\mathbb{Z}} \\ x & \mapsto & y \text{ such that } \forall i \in \mathbb{Z}, y_{[hi,h(i+1)[} = x_i \ . \end{array}$$

We need to be able to perform this encoding locally, we add some constraints to the alphabet B. Indeed, define the twosided subshift $\Lambda = \mathcal{O}_\sigma(\boxplus_h(B^{\mathbb{Z}})) = \bigcup_{0\leq i<h} \sigma^i(\boxplus_h(B^{\mathbb{Z}}))$. We want this union to be disjoint, in order to know, for any configuration of Λ, up to which shift it can be considered a sequence of macrocells. For this purpose, we add a *freezing* condition to B as follows.

Freezingness. A set $W \subset A^h$ is *p-freezing*, with $p, h \in \mathbb{N}$, if $\forall i \in [1, p]\,, A^i W \cap W A^i \neq \emptyset$, *i.e.* words from W cannot overlap on $h - p$ letters or more.

When p is sufficiently large, we obtain the following property.

Proposition 3.2. *Let $W \subset A^h$ be $\left\lfloor \frac{h}{2} \right\rfloor$-freezing, with $h \in \mathbb{N}$. Then W^2 is $(h-1)$-freezing; $\Lambda = \bigcup_{0\leq i<h} \sigma^i(\boxplus_h(W^{\mathbb{Z}}))$ is a disjoint union and an SFT.*

If G is a CA of radius 1 on alphabet $B \subset A^h$, we can define its *h-ungrouped* partial CA $\boxtimes_h G$ on the subshift $\Lambda = \mathcal{O}_\sigma(\boxplus_h(B^{\mathbb{Z}}))$, of radius $2h-1$ and local rule:

$$f : \begin{array}{rcl} \mathcal{L}_{4h-1}(\Lambda) & \to & A \\ w & \mapsto & g(u^{-1}, u^0, u^1)_i \text{ if } \left\{ \begin{array}{l} w \in A^{h-1-i} u^{-1} u^0 u^1 A^i \\ u^{-1}, u^0, u^1 \in B \\ i \in [0, h[\ . \end{array} \right. \end{array}$$

Proposition 3.3. *Let $B \subset A^h$ be $\left\lfloor \frac{h}{2} \right\rfloor$-freezing, and G a CA on alphabet B, of radius 1 and local rule $g : A^3 \to A$. Then the ungrouped CA $\boxtimes_h G$ is well defined and its trace is $\overset{\circ}{\tau}_G$.*

Proof. The local rule f as defined above is not ambiguous since the shift i is unique by Proposition 3.2. By construction, $f(A^{h-1}u^{-1}u^0u^1A^{h-1}) = g(u^{-1}u^0u^1)$, hence by a recurrence on $j \in \mathbb{N}$, we see that if $i \in [0, h[$ and $x \in \sigma^i(\boxplus_h(B^{\mathbb{Z}}))$, then $\forall k \in \mathbb{Z}, \boxtimes_h G^j(x)_0 = G^j((x_{[kh-i,(k+1)h-i[})_{i\in\mathbb{Z}})_i$. As a result, $\tau_{\boxtimes_h G} = \bigcup_{0\le i<h} \pi_i(\tau_G)$. ■

Borders. The freezing condition is very restrictive, but any alphabet can be modified in such way to satisfy this property, thanks to a suitable juxtaposition to some freezing set of words. Formally, a *border* for $B \subset A^k$, with $k \in \mathbb{N} \setminus \{0\}$, is a couple $(\Upsilon, \delta_\Upsilon)$, where $\Upsilon \subset A^l$ is $\lfloor \frac{k+l}{2} \rfloor$-freezing, and δ_Υ is a function from Υ into itself. From the latter, seen as the local rule, we define the CA $\Delta_\Upsilon : \Upsilon^{\mathbb{Z}} \to \Upsilon^{\mathbb{Z}}$ of radius 0 whose polytrace is $\bigcup_{0\le i<l} \pi_i(\mathcal{O}_{\delta_\Upsilon})$.

Borders will be used to separate words representing letters of B in an non-ambiguous way.

Proposition 3.4. *Let G be a CA on alphabet $B \subset A^k$ and $(\Upsilon \subset A^l, \delta_\Upsilon)$ a border for B. Then, the ungrouped CA $F = \boxtimes_{k+l}(\Delta_\Upsilon \times G)$ on the SFT $\Lambda = \mathcal{O}_\sigma({}^\infty(\Upsilon B)^\infty)$ is well defined and its trace is $\overset{\circ}{\tau}_G \cup \overset{\circ}{\tau}_{\Delta_\Upsilon}$.*

Proof. If $\Upsilon \subset A^l$ is $\lfloor \frac{k+l}{2} \rfloor$-freezing, then we can see that so is ΥB. Hence, Proposition 3.3 can be applied to $\Delta_\Upsilon \times G$, seen as a CA on alphabet ΥB. ■

In the following, we describe a first example of borders.

Corollary 3.5. *Let Σ be a polytraceable subshift which contains two distinct uniform infinite words 0^∞ et 1^∞. Then, Σ is partially traceable effectively from a polytracing CA and these two words.*

Proof. Define $\Upsilon^k_{(0,1)} = \{10^k\}$. Note that $\Upsilon^k_{(0,1)}$ is k-freezing so $(\Upsilon^k_{(0,1)}, \mathrm{id})$ is a border. Applying Proposition 3.4, as $\overset{\circ}{\tau}_{\Delta_{\Upsilon^k_{(0,1)}}} = \{0^\infty, 1^\infty\}$, we get that Σ is partially traceable. ■

Dynamical borders. In the case where the polytraceable subshift does not contain two uniform infinite words, we must find another condition to get a freezing alphabet. Assume it contains some periodic non-uniform infinite word u^∞. We note $\overline{u} = u_{|u|-1} \dots u_0$ the reverse of u and $\gamma^i(u)$ the i^{th} rotation $u_{[i,|u|[}u_{[0,i[}$ of u, for $0 \le i < |u|$. Then the following represents a border: let $\Upsilon^k_u = \left\{ u_i^{k+3|u|}\overline{\gamma^i(u)}\gamma^i(u)u_i^{|u|} \middle| 0 \le i < |u| \right\} \subset A^{k+6|u|}$, and $\delta_{\Upsilon^k_u} : a^{k+3|u|}v\overline{v}a^{|u|} \mapsto v_1^{k+3|u|}\gamma(v)\overline{\gamma(v)}v_1^{|u|}$.

Proposition 3.6 ([CFG10]). Υ^k_u *is $(k + 3\,|u|)$-freezing.*

Corollary 3.7. *Let Σ be a polytraceable subshift which contains a periodic infinite word ${}^\infty u^\infty$ of smallest period $|u| > 1$. Then, Σ is partially traceable effectively from a polytracing CA and u.*

Proof. It is sufficient to apply Proposition 3.4 to the border $(\Upsilon^k_u, \delta_{\Upsilon^k_u})$. We can see that $\overset{\circ}{\tau}_{\Delta_{\Upsilon^k_u}} = \mathcal{O}_\sigma(u^\infty)$, which allows to obtain a CA $F : \Lambda \to \Lambda$ such that $\tau_F = \overset{\circ}{\tau}_G$. ■

Actually, the only sofic subshifts which are not concerned by the two previous constructions are the nilpotent ones.

Lemma 3.8 ([CFG10])**.** *Any nilpotent subshift is partially traceable effectively from the subshift.*

The following gives an example of subshift which is nilpotent, hence partially traceable, but not traceable.

Example 3.9 ([CFG10])**.** No CA traces the subshift $\mathcal{O}_\sigma((\lambda + 1 + 01 + 001 + 21)0^\infty)$.

Putting things together, we get the following important results.

Proposition 3.10. *Any polytraceable sofic subshift is partially traceable effectively from a polytracing CA.*

Proof. It is known that any sofic subshift Σ admits some periodic infinite word u^∞, and that it is unique only if Σ is weakly nilpotent. In this case, as the projection of some trace, it is nilpotent by Theorem 2.3, and Lemma 3.8 allows to conclude. If there are several distinct periodic infinite words among which one is non-uniform, then we can apply Corollary 3.7; otherwise there are several uniform periodic words and we can apply Corollary 3.5. ■

The previous proposition, together with Theorem 2.5, gives the following – note that the SFT are partially traceable directly from the definition.

Corollary 3.11. *Any uncountable sofic subshift is partially traceable effectively from it.*

4. Ultimate traceability

In this section we consider traces of CA up to ultimate coincidence, *i.e.* assimilating any two subshifts that are different in only a finite number of cells.

One of the difficulties in making traces (Theorem 2.7), avoided in partial traces, was to deal with "invalid" configurations, not in $\mathcal{O}_\sigma(\boxplus_h(B^{\mathbb{Z}}))$. At location of "errors" (*i.e.* sites where a pattern of the configuration is not a pattern of $\boxplus_h(B^{\mathbb{Z}})$), instead of applying the simulating rule, we apply a default rule. However, once one of these rules is chosen, the cell must keep using it forever in order to stay in the "right" subshift.

The possibility of initially altering some cells of the subshift simplifies the problem. Indeed, it allows us to build borders in one round and remove all the "errors" in the initial configuration. We say that two subshifts Γ and Σ *ultimately coincide* if there exists some generation $J \in \mathbb{N}$ such that $\sigma^J(\Gamma) = \sigma^J(\Sigma)$.

Definition 4.1 (Ultimately traceable)**.** A subshift Σ is *ultimately traceable* if there is a CA G such that τ_G ultimately coincides with Σ. If F and J can be computed effectively from data D, we say that Σ is ultimately traceable *effectively from* D.

Note that any ultimately traceable subhift is a subsystem of some traceable subshift, and by Proposition 2.2 contains some deterministic subshift, but which may not involve all the letters of the alphabet.

Example 4.2. Consider the subshift $\Sigma = \mathcal{O}_\sigma((001)^\infty)$. It is an SFT. It is thus polytraceable, but not ultimately traceable since it does not admit any deterministic subshift.

The proof of the following proposition can be found in the online version.

Proposition 4.3 ([CFG10])**.** *Let $\Sigma \subset A^{\mathbb{N}}$ be a totally polytraceable subshift which contains some non-nilpotent deterministic subshift $\mathcal{O}_\xi$, $\xi : A \to A$. Then Σ is traceable effectively from a polytracing CA and ξ.*

With respect to Lemma 2.6 two additional hypotheses – first, that the subshift is totally polytraceable and, second, that the deterministic subshift is not nilpotent – help get rid of the complex CDD condition, and therefore to get a more precise result about ultimate traces.

Lemma 4.4. *If Σ is a polytraceable subshift, then there exists a subshift $\tilde{\Sigma}$ such that $\sigma(\Sigma) = \sigma(\tilde{\Sigma})$, totally polytraceable effectively from a polytracing CA.*

Proof. Let G be a CA polytracing Σ. Let $\psi : A^k \to B$ be a projection such that $\psi_{|B} = \mathrm{id}$; it can be seen as the local rule of some CA Ψ of radius 0. Define $\tilde{G} = G\Psi$. By construction, we can see that $\tilde{G}_{|B^{\mathbb{Z}}} = G$ and that $\tilde{G}((A^k)^{\mathbb{Z}}) = G(B^{\mathbb{Z}}) \subset B^{\mathbb{Z}}$, *i.e.* since the second time step the two traces coincide. ■

Proposition 4.5. *Let $\Sigma \subset A^{\mathbb{N}}$ be a polytraceable sofic subshift that contains some deterministic subshift $\mathcal{O}_\xi$, with $\xi : A' \to A'$ and $A' \subset A$. Then Σ ultimately coincides with some subshift $\tilde{\Sigma}$ which is traceable effectively from a polytracing CA, Σ and ξ.*

Proof. Let G be a CA on $B \subset A^k$ polytracing Σ, $k \in \mathbb{N} \setminus \{0\}$. Should we replace Σ by the corresponding $\tilde{\Sigma}$ of Lemma 4.4, we can assume that $B = A^k$.

- If Σ is weakly nilpotent, then, by Theorem 2.3, it is nilpotent, *i.e.* there is some $J \in \mathbb{N}$ such that $\sigma^J(\Sigma) = \{^\infty 0^\infty\}$, property which can be effectively tested from Σ; any nilpotent CA has a trace which ultimately coincides.
- If $\mathcal{O}_\xi$ is not nilpotent, then Proposition 4.3 can be applied to build a CA whose trace will be the polytrace of G.
- Suppose $\mathcal{O}_\xi$ is nilpotent, *i.e.* there is some $J \in \mathbb{N}$ and some state $0 \in A$ such that $\xi^J(A') = \{0\}$; we define:
$$\xi' : \begin{array}{ccc} A & \to & A \\ a & \mapsto & 0 \end{array} .$$
Since the trace $\tau_{\tilde{G}}$ is not weakly nilpotent, it contains some periodic infinite word w^∞, with $w \in A^+ \setminus 0^+ = A^+ \setminus \xi'(A)^+$. Hence, we can apply Lemma 2.6 to build a CA $\tilde{G} : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ such that $\tau_{\tilde{G}} = \mathring{\tau}_G \cup \mathcal{O}_{\xi'}$. As a result, $\sigma(\tau_{\tilde{G}}) = \sigma(\mathring{\tau}_G) \cup \{^\infty 0^\infty\} = \sigma(\mathring{\tau}_G)$. ■

Corollary 4.6. *Any SFT containing some deterministic subshift and any uncountable sofic subshift containing some deterministic subshift is ultimately traceable effectively from it.*

Here is an example of subshift which is not traceable, but ultimately traceable.

Example 4.7 ([CFG07])**.** The subshift $\Sigma = \{0^\infty, (01)^\infty, (10)^\infty\}$ is an SFT and contains some deterministic subshift, but is not traceable.

The previous corollary is not an equivalence: there are countable sofic ultimately traceable subshifts which are not SFT.

Example 4.8 ([CFG07])**.** The subshift $(0^*1 + 1^*)0^\infty$ is sofic, numerable, of infinite type, but traceable.

The study of the ultimate trace of some CA F is related to that of the limit trace, that is the set $\bigcap_{j\in\mathbb{N}}\sigma^j(\tau_F)$ of traces of configurations which can appear arbitrarily late. In particular, we can see that a surjective subshift which ultimately coincides with the trace of some CA is its limit trace. If it is sofic, the converse is true.

The bitrace of some CA F is the set of its "biorbits":

$$\tau_F^* = \left\{ (x_0^j)_{j\in\mathbb{Z}} \middle| \forall j \in \mathbb{Z}, x^j \in A^{\mathbb{Z}} \text{ and } F(x^j) = x^{j+1} \right\}.$$

We can see that it is the twosided subshift with the same language than the limit trace. As a consequence, we get the following.

Corollary 4.9. *Any onesided surjective subshift containing some deterministic subshift which is either of finite type or uncountable sofic is the limit trace of some stable CA. Any twosided subshift containing some deterministic subshift which is either of finite type or uncountable sofic is the bitrace of some stable CA.*

5. Undecidability

Let F a CA of diameter d, anchor m, local rule f on alphabet A. A state $0 \in A$ is 0-*spreading* if $d > 1$ and for all $u \in A^d$ such that $0 \sqsubset u$, we have $f(u) = 0$. The CA F is *spreading* if it is s-spreading for some $s \in A$.

The CA F is 0-*nilpotent* (or simply *nilpotent*) if there exists a $J > 0$ such that $F^J(A^{\mathbb{Z}}) = {}^\infty 0^\infty$. The proof technique developed in [Kar92] allows to prove the following.

Theorem 5.1. *The problem whether a spreading CA F is nilpotent is undecidable.*

In the sequel, we use the spreading state to control the evolution of another CA, generalizing the construction used in [CG07].

Consider two CA F_1 and F_2 of local rules f_1 and f_2 on (disjoint) alphabets A_1 and A_2. Without loss of generality, assume that they have the same diameter d and anchor m. Let $A = A_1 \cup A_2$ and $\varphi : A \to A_1$ a projection such that $\varphi_{|A_1} = \text{id}$. Let N and N_2 be two CA with the same diameter d and anchor m, local rules n, n_2, and alphabets B and $B_2 \subset B$, with $0 \in B_2$ being spreading for N_2. We build the CA H of same diameter d and anchor m, alphabet $A \times B$ and local rule:

$$h: \begin{array}{rcl} (A\times B)^d & \to & A \times B \\ (a_i, b_i)_{-m\le i<d-m} & \mapsto & \left| \begin{array}{ll} (f_2(a), n_2(b)) & \text{if } a \in A_2^d \text{ and } b \in (B_2 \setminus \{0\})^d \;, \\ (f_1 \circ \varphi(a), n(b)) & \text{otherwise .} \end{array} \right. \end{array}$$

Starting from a configuration in $(A_2 \times B_2)^{\mathbb{Z}}$, the CA simulates independently F_2 and N_2 (first part of the rule) until one 0 appears; at that moment they both change their rules; this change can happen only once for each cell, since from then the letters of the left component remain in A_1; hence the two components simulate F_1 and N respectively (second part).

The following notions and lemma will help us understand the dynamics of this CA. A set $U \subset A^k$, with $k \in \mathbb{N}\setminus\{0\}$ is *spreading* if $F([U]_1) \subset [U]_0 \cap [U]_1$ or $F([U]_0) \subset [U]_0 \cap [U]_1$. If F is a CA on alphabet A and $A' \subset A$, then we say that F is (globally) A'-*mortal* if $\forall x \in A^{\mathbb{Z}}, \exists i \in \mathbb{Z}, \exists j \in \mathbb{N}, F^j(x)_i \in A'$.

Lemma 5.2. *If F is a CA on alphabet A and $A' \subset A$ is spreading, then F is A'-mortal if and only if $\exists J \in \mathbb{N}, \forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \forall j \ge J, F^j(x)_i \in A'$.*

Proof. Suppose F is A'-mortal. By compacity, there is some $J \in \mathbb{N}$ and some radius $I \in \mathbb{N}$ such that $\forall x \in A^{\mathbb{Z}}, \exists i \in [-I, I], F^J(x)_i \in A'$. If A' is left-spreading, we obtain thanks to a trivial recurrence, $\forall x \in A^{\mathbb{Z}}, F^{J+2I}(x)_{-I} \in A'$. Thanks to uniformity and shift-invariance, we obtain the stated result. The right-spreading case is symetric. ∎

Lemma 5.3.

- *If N_2 is nilpotent, then there is some $J \in \mathbb{N}$ such that $\pi_0(H^J((A \times B)^{\mathbb{Z}})) \subset A_1^{\mathbb{Z}}$ and then, on $H^J((A \times B)^{\mathbb{Z}})$, H behaves like $F_1 \times N$.*
- *Otherwise, there is a subshift $\Lambda \subset B_2^{\mathbb{Z}}$ such that $\pi_0 \circ H_{|A_2^{\mathbb{Z}} \times \Lambda} = F_2 \circ \pi_0$.*

Proof.

- Suppose N_2 is nilpotent. From the definition of H, no orbit implies always the first part of the rule: H is $A_1 \times B$-mortal. Moreover we can see that $A_1 \times B$ is spreading for H. Thanks to Lemma 5.2, H remains ultimately on the alphabet $A_1 \times B$.
- Otherwise, there exists, thanks to Lemma 5.2, some configuration $x \in B_2^{\mathbb{Z}}$ such that $\forall i \in \mathbb{Z}, \forall j \in \mathbb{N}, N_2^j(x)_i \neq 0$; the subshift $\Lambda = \overline{\mathcal{O}_\sigma(\mathcal{O}_N(x))}$ is such that $A_2^{\mathbb{Z}} \times \Lambda$ is H-invariant and its first column is F_2. ∎

Since they are reduced to the nilpotency of the spreading CA N_2, the two cases presented are recursively inseparable, provided that they are disjoint.

Properties of ultimate polytraces. As for the conditions of traceability, polytraces represent here a useful intermediary tool.

Let G a CA on alphabet $\{0,1\}$ and N a CA on alphabet $\{0,1\}$ of radius 0 and locale rule $\xi : \{0,1\} \to \{0,1\}$ such that $\mathcal{O}_\xi \subset \tau_G$. We build the alphabets $A_1 = \{\,(a,a,b)|\, a,b \in \{0,1\}\,\}$ and $A_2 = \{0,1\}^3 \setminus A_1$, as well as the CA $F_1 = (N \times N \times G)_{|A_1}$, $F_2 = (\sigma \times \sigma \times G)_{|A_2}$. We can apply Lemma 5.3 to the CA H built as above from F_1, F_2, N, and any 0-spreading CA N_2 on alphabet $\{0,1\}$.

The product is here composed of four layers. The fourth one controls the whole behavior thanks to its spreading state 0. The third one simulates G independently. When the two first ones are distinct, they simulate full shifts (whose trace is $\{0,1\}^{\mathbb{N}}$) that hide the trace of G. As soon as some 0 appears in the last layer, they stop, unify and then apply ξ, which is contained in τ_G.

In the end of the section, we consider that H is built from G, N and N_2, the CA F_1 and F_2 being defined as above.

Lemma 5.4.

- *If N_2 is nilpotent, then $\overset{\circ}{\tau}_H$ ultimately coincides with τ_G.*
- *Otherwise, $\overset{\circ}{\tau}_H = \{0,1\}^{\mathbb{N}}$.*

Proof.

- Thanks to Lemma 5.3, if N_2 is nilpotent, then the first three components of H and $(N \times N \times G)_{|A_1}$ ultimately coincide, the trace of the last component being ultimately included in τ_N. Considering that the polytrace of $(N \times N \times G)_{|A_1}$ is $\tau_G \cup \tau_N$ and that, by hypothesis, $\tau_N \subset \tau_G$ the polytrace of H ultimately coincides with τ_G.
- Otherwise, there exists a subshift Λ such that the partial CA $H_{|A_2^{\mathbb{Z}} \times \Lambda}$ admits as first three projections $(\sigma \times \sigma \times G)_{|A_2^{\mathbb{Z}}}$. The first projection of the trace is $\{0,1\}^{\mathbb{N}}$, since for any infinite word a, there is another word b distinct in every cell ($\forall i \in \mathbb{N}$, $a_i \neq b_i$); hence the trace τ_H contains and therefore is $\{0,1\}^{\mathbb{N}}$. ∎

Properties of traces. As in the previous section, we are now going to simulate CA on alphabets with several components to transform the result on polytraces into a result on traces.

Lemma 5.5. *Let G a non-nilpotent onesided CA whose trace is not $\{0,1\}^{\mathbb{N}}$. The set of CA on alphabet $\{0,1\}$ whose trace is $\{0,1\}^{\mathbb{N}}$ is recursively inseparable from the set of CA on alphabet $\{0,1\}$ whose trace ultimately coincides with τ_G.*

Proof. Let N_2 a onesided 0-spreading CA.

- Suppose that the trace τ_G contains some non-nilpotent deterministic subshift $\mathcal{O}_\xi$, with $\xi : \{0,1\} \to \{0,1\}$. ξ can be seen as the local rule of the CA N. Build CA H as before. From Proposition 4.3, H can be transformed into some CA F on alphabet $\{0,1\}$ such that $\tau_F \overset{\circ}{=} \tau_H$.
- If the trace τ_G does not contain any non-nilpotent deterministic subshift, then, as it is still non-nilpotent, it contains some periodic infinite word w^∞, $w \in \{0,1\}^*$, $w \notin 0^*$. We can define the null CA $N = \overline{0}$ on $\{0,1\}^{\mathbb{N}}$ of local rule $\xi' : a \mapsto 0$ and define H as before. Remark that w^∞ and 0^∞ are in the trace of H, hence we can apply Lemma 2.6 to build a CA F on alphabet $\{0,1\}$ such that $\tau_F \overset{\circ}{=} \tau_H$.

In both cases, Lemma 5.4 gives that if N_2 is 0-nilpotent, then τ_F ultimately coincides with τ_G, otherwise $\tau_F = \{0,1\}^{\mathbb{N}}$. As F is computable from G, were the two cases separable, Theorem 5.1 would be contradicted. ■

From the remark that some CA traces are not equal to the full shift, we can see that this behavior is undecidable. But the previous lemma also infers other nontrivial properties of traces.

A property $\mathcal{P}$ over subshifts is *stable by ultimate coincidence* if for any subshifts Σ and Γ which ultimately coincide, we have $\Sigma \in \mathcal{P} \iff \Gamma \in \mathcal{P}$.

Theorem 5.6. *Let $\mathcal{P}$ be a property over subshifts which:*

(1) is satisfied by the trace subshift of some CA over alphabet $\{0,1\}$, but not all;
(2) is stable by ultimate coincidence.

Then, the problem

Instance: *a CA G on alphabet $\{0,1\}$.*
Question: *does τ_G satisfy property $\mathcal{P}$?*

is undecidable.

Proof. Let $\mathcal{P}$ be such a property and assume that $\{0,1\}^{\mathbb{N}}$ does not satisfy $\mathcal{P}$, should we take the complement. If $\mathcal{P}$ is only satisfied by nilpotent subshifts, then thanks to stability by ultimate coincidence, it is equivalent either to 0-nilpotency, to 1-nilpotency or to nilpotency, which are all undecidable by Theorem 5.1. Otherwise, $\mathcal{P}$ is satisfied by the trace τ_G of some non-nilpotent CA G. Would an algorithm decide $\mathcal{P}$, it would allow to separate the trace τ_G to $\{0,1\}^{\mathbb{N}}$ among traces over alphabet $\{0,1\}$ up to ultimate coincidence, contradicting Lemma 5.5. ■

This result includes in particular the so-called "nilpotent-stable" properties defined in [CG07], such as fullness, finiteness, ultimate periodicity, soficness, finite type, inclusion of a particular word as a factor. It also includes nilpotency, as well as all properties of the trace of the limit system $(\bigcap_{J \in \mathbb{N}} F^j(A^{\mathbb{Z}}), F)$ of CA F, as stated in [Gui08]. Moreover, it can be easily adapted to larger traces, *i.e.* taking the states of a central group of cells of each configuration. We can also see that this theorem implies the undecidability of all properties of any line projection of two-dimensional SFT (tilings respecting local constraints).

6. Conclusions

In our study of CA traces, we have reached two kinds of important results. On the one hand, we provided sufficient conditions for a subshift to be a polytrace, a trace, a partial trace, an ultimate trace. On the other hand, we proved the undecidability of nearly all properties over ultimate traces. Going beyond undecidability, when it is clear that the trace has been generated by CA, it would be interesting to study which ones, and with which minimal radius.

Remark that the contructions used in the paper build CA with a very large radius. It would be interesting to study the traces produced by cellular automata of a given fixed radius. This is not a so great limitation in complexity, since elementary CA (binary alphabet, radius 1) already present rich different behaviors. In particular, a deeper study of the so-called "canonical factors", *i.e.* traces which width is the radius of the CA, could be fundamental to fully understand this notion.

Another interesting research direction consists in trying to adapt or find some refinement of Kůrka's language classification ([Kůr97]) to the case of traces or ultimate traces. This would provide an interesting link between the complexity of the dynamics of CA and the (language) complexity of its traces.

References

[CFG07] Julien Cervelle, Enrico Formenti, and Pierre Guillon. Sofic trace of a cellular automaton. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *Computation and Logic in the Real World, 3^{rd} Conference on Computability in Europe (CiE07)*, volume 4497 of *Lecture Notes in Computer Science*, pages 152–161, Siena, Italy, June 2007. Springer-Verlag.

[CFG10] Julien Cervelle, Enrico Formenti, and Pierre Guillon. Ultimate trace of cellular automata. Technical report, http://arxiv.org/abs/1001.0251, January 2010.

[CG07] Julien Cervelle and Pierre Guillon. Towards a Rice theorem on traces of cellular automata. In Ludek Kučera and Antonín Kučera, editors, *32^{nd} International Symposium on the Mathematical Foundations of Computer Science*, volume 4708 of *LNCS*, pages 310–319, Český Krumlov, Czech Republic, august 2007. Springer-Verlag.

[GR08] Pierre Guillon and Gaétan Richard. Nilpotency and limit sets of cellular automata. In Edward Ochmański and Jerzy Tyszkiewicz, editors, *33^{rd} International Symposium on the Mathematical Foundations of Computer Science (MFCS'08)*, volume 5162 of *LNCS*, pages 375–386, Toruń, Poland, august 2008. Springer-Verlag.

[Gui08] Pierre Guillon. *Automates cellulaires : dynamiques, simulations, traces*. PhD thesis, Université Paris-Est, November 2008.

[Hed69] Gustav Arnold Hedlund. Endomorphism and automorphism of the shift dynamical system. *Math. Sys. Theory*, 3:320–375, 1969.

[Kar92] Jarkko Kari. The nilpotency problem of one-dimensional cellular automata. *SIAM J. on Computing*, 21(3):571–586, 1992.

[Kůr97] Petr Kůrka. Languages, equicontinuity and attractors in cellular automata. *Erg. Th. & Dyn. Sys.*, 17:417–433, 1997.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 167-178
www.stacs-conf.org

TWO-PHASE ALGORITHMS FOR THE PARAMETRIC SHORTEST PATH PROBLEM

SOURAV CHAKRABORTY[1] AND ELDAR FISCHER[2] AND ODED LACHISH[3] AND RAPHAEL YUSTER[4]

[1] CWI, Amsterdam, Netherlands
E-mail address: sourav.chakraborty@cwi.nl

[2] Department of Computer Science, Technion, Haifa 32000, Israel
E-mail address: eldar@cs.technion.ac.il

[3] Centre for Discrete Mathematics and its Applications, University of Warwick , Coventry, UK
E-mail address: oded@dcs.warwick.ac.uk

[4] Department of Mathematics, University of Haifa, Haifa 31905, Israel
E-mail address: raphy@math.haifa.ac.il

ABSTRACT. A *parametric weighted graph* is a graph whose edges are labeled with continuous real functions of a single common variable. For any instantiation of the variable, one obtains a standard edge-weighted graph. Parametric weighted graph problems are generalizations of weighted graph problems, and arise in various natural scenarios. Parametric weighted graph algorithms consist of two phases. A *preprocessing phase* whose input is a parametric weighted graph, and whose output is a data structure, the advice, that is later used by the *instantiation phase*, where a specific value for the variable is given. The instantiation phase outputs the solution to the (standard) weighted graph problem that arises from the instantiation. The goal is to have the running time of the instantiation phase supersede the running time of any algorithm that solves the weighted graph problem from scratch, by taking advantage of the advice.

In this paper we construct several parametric algorithms for the shortest path problem. For the case of linear function weights we present an algorithm for the single source shortest path problem. Its preprocessing phase runs in $\tilde{O}(V^4)$ time, while its instantiation phase runs in only $O(E + V \log V)$ time. The fastest standard algorithm for single source shortest path runs in $O(VE)$ time. For the case of weight functions defined by degree d polynomials, we present an algorithm with quasi-polynomial preprocessing time $O(V^{(1+\log f(d)) \log V})$ and instantiation time only $\tilde{O}(V)$. In fact, for any pair of vertices u, v, the instantiation phase computes the distance from u to v in only $O(\log^2 V)$ time. Finally, for linear function weights, we present a randomized algorithm whose preprocessing time is $\tilde{O}(V^{3.5})$ and so that for any pair of vertices u, v and any instantiation variable, the instantiation phase computes, in $O(1)$ time, a length of a path from u to v that is at most (additively) ϵ larger than the length of a shortest path. In particular, an all-pairs shortest path solution, up to an additive constant error, can be computed in $O(V^2)$ time.

1998 ACM Subject Classification: F.2.3.

Key words and phrases: Parametric Algorithms, Shortest path problem.

Part of the research done when the first author was a postdoc in Technion.

For the second author research was supported by an ERC-2007-StG grant number 202405 and by an ISF grant 1011/06.

Third author was supported in part by EPSRC award EP/G064679/1 and by the Centre for Discrete Mathematics and its Applications (DIMAP), EPSRC award EP/D063191/1.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2452

1. Introduction

In networking or telecommunications the search for the minimum-delay path (that is the shortest path between two points) is always on. The cost on each edge, that is the time taken for a signal to travel between two adjacent nodes of the network, is often a function of real time. Hence the shortest path between any two nodes changes with time. Of course one can run a shortest path algorithm every time a signal has to be sent, but usually some prior knowledge of the network graph is given in advance, such as the structure of the network graph and the cost functions on each edge (with time as a variable).

How can one benefit from this extra information? One plausible way is to preprocess the initial information and store the preprocessed information. Every time the rest of the input is given, using the preprocessed information, one can solve the optimization problem faster than solving the problem from scratch. Even if the preprocessing step is expensive one would benefit by saving precious time whenever the optimal solution has to be computed. Also, if the same preprocessed information is used multiple times then the total amount of resources used will be less in the long run.

Similar phenomena can be observed in various other combinatorial optimization problems that arise in practice; that is, a part of the input does not change with time and is known in advance. However, many times it is hard to make use of this extra information.

In this paper we consider only those problems where the whole input is a weighted graph. We assume that the graph structure and some knowledge of how the weights on the edges are generated are known in advance. We call this the *function-weighted graph* – it is a graph whose edges are labeled with continuous real functions. When all the functions are univariate (and all have the same variable), the graph is called a *parametric weighted graph*. In other words, the graph is $G = (V, E, W)$ where $W : E \to \mathcal{F}$ and $\mathcal{F}$ is the space of all real continuous functions with the variable x. If G is a parametric weighted graph, and $r \in \mathbb{R}$ is any real number, then $G(r)$ is the standard weighted graph where the weight of an edge e is defined to be $(W(e))(r)$. We say that $G(r)$ is an *instantiation* of G, since the variable x in each function is instantiated by the value r. Parametric weighted graphs are therefore, a generic instance of infinitely many instances of weighted graphs.

The idea is to use the generic instance G to precompute some general generic information $I(G)$, such that for any given instantiation $G(r)$, we will be able to use the precomputed information $I(G)$ in order to speed up the time to solve the given problem on $G(r)$, faster than just solving the problem on $G(r)$ from scratch. Let us make this notion more precise.

A *parametric weighted graph algorithm* (or, for brevity, a *parametric algorithm*) consists of two phases. A *preprocessing phase* whose input is a parametric weighted graph G, and whose output is a data structure (the advice) that is later used by the *instantiation phase*, where a specific value r for the variable is given. The instantiation phase outputs the solution to the (standard) weighted graph problem on the weighted graph $G(r)$. Naturally, the goal is to have the running time of the instantiation phase significantly smaller than the running time of any algorithm that solves the weighted graph problem from scratch, by taking advantage of the advice constructed in the preprocessing phase. Parametric algorithms are therefore evaluated by a pair of running times, the *preprocessing time* and the *instantiation time*.

In this paper we show that parametric algorithms are beneficial for one of the most natural combinatorial optimization problems: the *shortest path* problem in directed graphs. Recall that given a directed real-weighted graph G, and two vertices u, v of G, the distance from u to v, denoted by $\delta(u, v)$, is the length of a shortest path from u to v. The *single pair* shortest path problem seeks to compute $\delta(u, v)$ and construct a shortest path from u to v. Likewise, the *single source* shortest path problem seeks to compute the distances and shortest paths from a given vertex to all other vertices, and the *all pairs* version seeks to compute distances and shortest paths between all ordered pairs of vertices. In some of our algorithms we forgo the calculation of the path itself to achieve a shorter instantiation time. In all those cases the algorithms can be easily modified to also output a shortest path, in which case their instantiation time is the sum of the time it takes to calculate the distance and a time linear in the size of the path to be output.

Our first algorithm is a parametric algorithm for single source shortest path, in the case where the weights are *linear* functions. That is, each edge e is labeled with a function $a_e x + b_e$ where a_e and b_e are reals. Such linear parametrization has practical importance. Indeed, in many problems the cost of an edge is composed from some constant term plus a term which is a factor of some commodity, whose cost varies (e.g. bank commissions, taxi fares, vehicle maintenance costs, and so on). Our parametric algorithm has preprocessing time $\tilde{O}(n^4)$ and instantiation time $O(m + n \log n)$ (throughout this paper n and m denote the number of vertices and edges of a graph, respectively). We note that the fastest algorithm for the single source shortest path in real weighted directed graphs requires $O(nm)$ time; the Bellman-Ford algorithm [2]. The idea of our preprocessing stage is to precompute some other linear functions, on the *vertices*, so that for every instantiation r, one can quickly determine whether $G(r)$ has a negative cycle and otherwise use these functions to quickly produce a reweighing of the graph so as to obtain only nonnegative weights similar to the weights obtained by Johnson's algorithm [12]. In other words, we *avoid* the need to run the Bellman-Ford algorithm in the instantiation phase. The $\tilde{O}(n^4)$ time in the preprocessing phase comes from the use of Megiddo's[13] technique that we need in order to compute the linear vertex functions.

Theorem 1.1. *There exists a parametric algorithm for single source shortest path in graphs weighted by linear functions, whose preprocessing time is $\tilde{O}(n^4)$ and whose instantiation time is $O(m + n \log n)$.*

Our next algorithm applies to a more general setting where the weights are polynomials of degree at most d. Furthermore, in this case our goal is to have the instantiation phase answering distance queries between any two vertices in *sublinear* time. Notice first that if we allow exponential preprocessing time, this goal can be easily achieved. This is not hard to see since the overall possible number of shortest paths (when x varies over the reals) is $O(n!)$, or from Fredman's decision tree for shortest paths whose height is $O(n^{2.5})$ [8]. But can we settle for *sub-exponential* preprocessing time and still be able to have sublinear instantiation time? Our next result achieves this goal.

Theorem 1.2. *There exists a parametric algorithm for the single pair shortest path problem in graphs weighted by degree d polynomials, whose preprocessing time is $O(n^{(O(1)+\log f(d)) \log n})$ and instantiation time $O(\log^2 n)$, where $f(d)$ is the time required to compute the intersection points of two degree d polynomials. The size of the advice that the preprocessing algorithm produces is $O(n^{(O(1)+\log d) \log n})$.*

The above result falls in the subject of sensitivity analysis where one is interested in studying the effect on the optimal solution as the value of the parameter changes. We give a linear-time (linear in the output size) algorithm that computes the breaking points.

The practical and theoretical importance of shortest path problems lead several researchers to consider fast algorithms that settle for an approximate shortest path. For the general case (of real weighted digraphs) most of the algorithms guarantee an α-*stretch* factor. Namely, they compute a path whose length is at most $\alpha\delta(u, v)$. We mention here the $(1 + \epsilon)$-stretch algorithm of Zwick for the all-pairs shortest path problem, that runs in $\tilde{O}(n^{\omega})$ time when the weights are non-negative reals [18]. Here $\omega < 2.376$ is the matrix multiplication exponent [5].

Here we consider probabilistic additive-approximation algorithms, or *surplus* algorithms, that work for linear weights which may have positive and negative values (as long as there is no negative weight cycle). We say that a shortest path algorithm has an ϵ-surplus if it computes paths whose lengths are at most $\delta(u, v) + \epsilon$. We are unaware of any truly subcubic algorithm that guarantees an ϵ-surplus approximation, and which outperforms the fastest general all-pairs shortest path algorithm [4].

In the linear-parametric setting, it is easy to obtain ϵ-surplus parametric algorithms whose preprocessing time is $O(n^4)$ time, and whose instantiation time, for any ordered pair of queried vertices u, v is constant. It is assumed instantiations are taken from some interval I whose length is independent of n. Indeed, we can partition I into $O(n)$ subintervals $I_1, I_2, \ldots$ of size $O(1/n)$ each, and solve, in cubic time (say, using [7]), the exact all-pairs solution for any instantiation r that is an endpoint of two consecutive intervals. Then, given any $r \in I_j = (a_j, b_j)$, we simply look at the solution for b_j and notice that we are (additively) off from the right answer only by $O(1)$. Standard scaling arguments can make the surplus smaller than ϵ. But do we really need to spend $O(n^4)$ time for preprocessing? In other words, can we invest (significantly) less than $O(n^4)$ time and still be able to answer instantiated distance queries in $O(1)$ time? The following result gives a positive answer to this question.

Theorem 1.3. *Let $\epsilon > 0$, let $[\alpha, \beta]$ be any fixed interval and let γ be a fixed constant. Suppose G is a linear-parametric graph that has no negative weight cycles in the interval $[\alpha, \beta]$, and for which every edge weight $a_e + xb_e$ satisfies $|a_e| \leq \gamma$. There is a parametric randomized algorithm for the ϵ-surplus shortest path problem, whose preprocessing time is $\tilde{O}(n^{3.5})$ and whose instantiation time is $O(1)$ for a single pair, and hence $O(n^2)$ for all pairs.*

We note that this algorithm works in the restricted addition-comparison model. We also note that given an ordered pair u, v and $r \in [\alpha, \beta]$, the algorithm outputs, in $O(1)$ time, a weight of an actual path from u to v in $G(r)$, and points to a linked list representing that path. Naturally, if one wants to output the vertices of this path then the time for this is linear in the length of the path.

The rest of this paper is organized as follows. The next subsection shortly surveys related research on parametric shortest path problems. In the three sections following it we prove Theorems 1.1, 1.2 and 1.3. Section 5 contains some concluding remarks and open problems.

1.1. Related research

Several researchers have considered parametric versions of combinatorial optimization problems. In particular function-weighted graphs (under different names) have been extensively studied in the subject of sensitivity analysis (see [11]) where they study the effect on the optimal solution as the parameter value changes.

Murty [14] showed that for parametric linear programming problems the optimal solution can change exponentially many times (exponential in the number of variables). Subsequently, Carstensen [3] has shown that there are constructions for which the number of shortest path changes while x varies over the reals is $n^{\Omega(\log n)}$. In fact, in her example each linear function is of the form $a_e + xb_e$ and both a_e and b_e are positive, and x varies in $[0, \infty]$. Carstensen also proved that this is tight. In other words, for any linear-parametric graph the number of changes in the shortest paths is $n^{O(\log n)}$. A simpler proof was obtained by Nikolova et al. [16], that also supply an $n^{O(\log n)}$ time algorithm to compute the path breakpoints. Their method, however, does not apply to the case where the functions are not linear, such as in the case of degree d polynomials. Gusfield [10] also gave a proof for the upper bound of the number of breakpoints in the linear function version of the parametric shortest path problem, in addition to studying a number of other parametric problems.

Karp and Orlin [15], and, later, Young, Tarjan, and Orlin [17] considered a special case of the linear-parametric shortest path problem. In their case, each edge weight e is either some fixed constant b_e or is of the form $b_e - x$. It is not too difficult to prove that for any given vertex v, when x varies from $-\infty$ to the largest x_0 for which $G(x_0)$ has no negative weight cycle (possibly $x_0 = \infty$), then there are at most $O(n^2)$ distinct shortest path trees from v to all other vertices. Namely, for each $r \in [-\infty, x_0]$ one of the trees in this family is a solution for single-source shortest path in $G(r)$. The results in [15, 17] cleverly and compactly compute all these trees, and the latter does it in $O(nm + n^2 \log n)$ time.

2. Proof of Theorem 1.1

The proof of Theorem 1.1 follows from the following two lemmas.

Lemma 2.1. *Given a linear-weighted graph $G = (V, E, W)$, there exist $\alpha, \beta \in \mathbb{R} \cup \{-\infty\} \cup \{+\infty\}$ such that $G(r)$ has no negative cycles if and only if $\alpha \leq r \leq \beta$. Moreover α and β can be found in $\tilde{O}(n^4)$ time.*

Lemma 2.2. *Let $G = (V, E, W)$ be a linear-weighted graph. Also let $\alpha, \beta \in \mathbb{R} \cup \{-\infty\} \cup \{+\infty\}$ be such that at least one of them is finite and for all $\alpha \geq r \geq \beta$ the graph $G(r)$ has no negative cycle. Then for every vertex $v \in V$ there exists a linear function $g_v^{[\alpha,\beta]}$ such that if the new weight function W' is given by*

$$W'((u,v)) = W((u,v)) + g_u^{[\alpha,\beta]} - g_v^{[\alpha,\beta]}$$

then the new linear-weighted graph $G' = (V, E, W')$ has the property that for any real $\alpha \leq r \leq \beta$ all the edges in $G'(r)$ are non-negative. Moreover the functions $g_v^{[\alpha,\beta]}$ for all $v \in V$ can be found in $O(mn)$ time.

So given a linear-weighted graph G, we first use Lemma 2.1 to compute α and β. If at least one of α and β is finite then using Lemma 2.2 we compute the n linear functions $g_v^{[\alpha,\beta]}$, one for each $v \in V$. If $\alpha = -\infty$ and $\beta = +\infty$, then using Lemma 2.2 we compute the $2n$ linear functions $g_v^{[\alpha,0]}$ and $g_v^{[0,\beta]}$. These linear functions will be the advice that the preprocessing algorithm produces. The above lemmas guarantee us that the advice can be computed in time $\tilde{O}(n^4)$, that is the preprocessing time is $\tilde{O}(n^4)$.

Now when computing the single source shortest path problem from vertex v for the graph $G(r)$ our algorithm proceeds as follows:

(1) If $r < \alpha$ or $r > \beta$ output "$-\infty$" as there exists a negative cycle (such instances are considered invalid).

(2) If $\alpha \leq r \leq \beta$ and at least one of α or β is finite then compute $g_u(r)$ for all $u \in V$. Use these to re-weight the edges in the graph as in Johnson's algorithm [12]. If $\alpha = -\infty$ and $\beta = +\infty$ then if $r \leq 0$ compute $g_u^{[\alpha,0]}(r)$ for all $u \in V$ and if $r \geq 0$ compute $g_u^{[0,\beta]}(r)$ for all $u \in V$. Notice that after the reweighing we have an instance of $G'(r)$.

(3) Use Dijkstra's algorithm [6] to solve the single source shortest path problem in $G'(r)$. Dijkstra's algorithm applies since $G'(r)$ has no negative weight edges. The shortest paths tree returned by Dijkstra's algorithms applied to $G'(r)$ is also the shortest paths tree in $G(r)$. As

in Johnson's algorithm, we use the results $d'(v,u)$ of $G'(r)$ to deduce $d(v,u)$ in $G(r)$ since, by Lemma 2.2 $d(v,u) = d'(v,u) - g_v(r) + g_u(r)$.

The running time of the instantiation phase is dominated by the running time of Dijkstra's algorithm which is $O(m + n \log n)$ [9].

2.1. Proof of Lemma 2.1

Since the weight on the edges of the graph G are linear functions, we have that the weight of any directed cycle in the graph is also a linear function. Let $C_1, C_2, \ldots, C_T$ be the set of all directed cycles in the graph. The linear weight function of a cycle C_i will be denoted by wt(C_i). If wt(C_i) is not the constant function, then let γ_i be the real number for which the linear equation wt(C_i) evaluates to 0.

Let α and β be defined as follows:

$$\alpha = \max_i \{\gamma_i \mid \text{ wt}(C_i) \text{ has a positive slope}\}.$$

$$\beta = \min_i \{\gamma_i \mid \text{ wt}(C_i) \text{ has a negative slope}\}.$$

Note that if wt(C_i) has a positive slope then $\gamma_i = \min_x \{\text{wt}(C_i)(x) \geq 0\}$. Thus for all $x \geq \gamma_i$ the value of wt(C_i) evaluated at x is non-negative. So by definition for all $x \geq \alpha$ the value of the wt(C_i) is non-negative if the slope of wt(C_i) is positive, and for any $x < \alpha$ there exists a cycle C_i such that wt(C_i) has positive slope and wt$(C_i)(x)$ is negative. Similarly, for all $x \leq \beta$ the value of the wt(C_i) is non-negative if the slope of wt(C_i) is negative and for any $x > \beta$ there exists a cycle C_i such that wt(C_i) has negative slope and wt$(C_i)(x)$ is negative.

This proves the existence of α and β. There are, however, two bad cases that we wish to exclude. Notice that if $\alpha > \beta$ this means that for any evaluation at x, the resulting graph has a negative weight cycle. The same holds if there is some cycle for which wt(C_i) is constant and negative. Let us now show how α and β can be efficiently computed whenever these bad cases do not hold. Indeed, α is the solution to the following Linear Program (LP), which has a feasible solution if and only if the bad cases do not hold.

Minimize x under the constraints

$\forall i$**, wt**$(C_i)(x) \geq 0.$

This is an LP on one variable, but the number of constraints can be exponential. Using Megiddo's[13] technique for finding the minimum ratio cycles we can solve the linear-program in $O(n^4 \log n)$ steps.

2.2. Proof of Lemma 2.2

Let α and β be the two numbers such that for all $\alpha \leq r \leq \beta$ the graph $G(r)$ has no negative cycles and at least one of α and β is finite.

First let us consider the case when both α and β are finite. Recall that, given any number r, Johnson's algorithm associates a weight function $h^r : V \to \mathbb{R}$ such that, for any edge $(u,v) \in E$,

$$W_{(u,v)}(r) + h^r(u) - h^r(v) \geq 0.$$

(Johnson's algorithm computes this weight function by running the Bellman-Ford algorithm over $G(r)$). Define the weight function $g_v^{[\alpha,\beta]}$ as

$$g_v^{[\alpha,\beta]}(x) = \left(\frac{h^\beta(v) - h^\alpha(v)}{\beta - \alpha}\right) x + h^\alpha(v) - \left(\frac{h^\beta(v) - h^\alpha(v)}{\beta - \alpha}\right) \alpha \; .$$

This is actually the equation of the line joining $(\alpha, h^\alpha(v))$ and $(\beta, h^\beta(v))$ in $\mathbb{R}^2$.
Now we need to prove that for every $\alpha \leq r \leq \beta$ and for every $(u, v) \in V$,

$$W_{(u,v)}(r) + g_u^{[\alpha,\beta]}(r) - g_v^{[\alpha,\beta]}(r) \geq 0 \; .$$

Since $\alpha \leq r \leq \beta$, one can write $r = (1-\delta)\alpha + \delta\beta$ where $1 \geq \delta \geq 0$. Then for all $v \in V$,

$$g_v^{[\alpha,\beta]}(r) = (1-\delta)h^\alpha(v) + \delta h^\beta(v) \; .$$

Since $W_{(u,v)}(r)$ is a linear function we can write

$$W_{(u,v)}(r) = (1-\delta)W_{(u,v)}(\alpha) + \delta W_{(u,v)}(\beta) \; .$$

So after re-weighting the weight of the edge (u, v) is

$$(1-\delta)W_{(u,v)}(\alpha) + \delta W_{(u,v)}(\beta) + (1-\delta)h^\alpha(u) + \delta h^\beta(u) - (1-\delta)h^\alpha(v) - \delta h^\beta(v) \; .$$

Now this is non-negative as by the definition of h^β and h^α we know that both $W_{(u,v)}(\beta) + h^\beta(u) - h^\beta(v)$ and $W_{(u,v)}(\alpha) + h^\alpha(u) - h^\alpha(v)$ are non-negative.

We now consider the case when one of α or β is not finite. We will prove it for the case where $\beta = +\infty$. The case $\alpha = -\infty$ follows similarly. Consider the simple weighted graph $G_\infty = (V, E, W_\infty)$ where the weight function W_∞ is defined as: if the weight of the edge e is $W(e) = a_e x + b_e$ then $W_\infty(e) = a_e$.

We run the Johnson's algorithm on the graph G_∞. Let $h^\infty(v)$ denote the weight that Johnson's algorithm associates with the vertex v. Then define the weight function $g_v^{[\alpha,\infty]}$ as

$$g_v^{[\alpha,\infty]}(x) = h^\alpha(v) + (x - \alpha)h^\infty(v) \; .$$

We need to prove that for every $\alpha \leq r$ and for every $(u, v) \in V$,

$$W_{(u,v)}(r) + g_u^{[\alpha,\infty]}(r) - g_v^{[\alpha,\infty]}(r) = W_{(u,v)}(r) + h^\alpha(u) + (r-\alpha)h^\infty(u) - h^\alpha(v) - (r-\alpha)h^\infty(v) \geq 0 \; .$$

Let $r = \alpha + \delta$ where $\delta \geq 0$. By the linearity of W we can write $W_{(u,v)}(r) = W_{(u,v)}(\alpha) + \delta a_{(u,v)}$, where $W_{(u,v)}(r) = a_{(u,v)} r + b_{(u,v)}$. So the above inequality can be restated as

$$W_{(u,v)}(\alpha) + \delta a_{(u,v)} + h^\alpha(u) + \delta h^\infty(u) - h^\alpha(v) - \delta h^\infty(v) \geq 0 \; .$$

This now follows from the fact that both $W_{(u,v)}(\alpha) + h^\alpha(u) - h^\alpha(v)$ and $a_{(u,v)} + h^\infty(u) - h^\infty(v)$ are non-negative.

Since the running time of the reweighing part of Johnson's algorithm takes $O(mn)$ time, the overall running time of computing the functions $g_v^{[\alpha,\beta]}$ is $O(mn)$, as claimed.

3. Proof of Theorem 1.2

In this section we construct a parametric algorithm that computes the distance $\delta(u, v)$ between a given pair of vertices. If one is interested in the actual path realizing this distance, then it can be found with some extra book-keeping that we omit in the proof.

The processing algorithm will output the following advice: for any pair $(u, v) \in V \times V$ the advice consists of a set of $t + 2$ increasing real numbers $-\infty = b_0 < b_1 < \cdots < b_t < b_{t+1} = \infty$ and an ordered set of degree-d polynomials $p_0, p_1, \ldots, p_t$, such that for all $b_i \leq r \leq b_{i+1}$ the weight of a shortest path in $G(r)$ from u to v is $p_i(r)$. Note that each p_i corresponds to the weight of a path from u to v. Thus if we are interested in computing the exact path then we need to keep track of the path corresponding to each p_i.

Given r, the instantiation algorithm has to find the i such that $b_i \leq r \leq b_{i+1}$ and then output $p_i(r)$. So the output algorithm runs in time $O(\log t)$. To prove our result we need to show that for any $(u, v) \in V \times V$ we can find the advice in time $O(f(d)n)^{\log n}$. In particular this will prove that $t = O(dn)^{\log n}$ and hence the result will follow.

Definition 3.1. A *minBase* is a sequence of increasing real numbers $-\infty = b_0 < b_1 < \cdots < b_t < b_{t+1} = \infty$ and an ordered set of degree-d polynomials $p_0, p_1, \ldots, p_t$, such that for all $b_i \leq r \leq b_{i+1}$ and all $j \neq i$, $p_i(r) \leq p_j(r)$.

We call the sequence of real numbers the *breaks*. We call each interval $[b_i, b_{i+1}]$ the i-th interval of the minBase and the polynomial p_i the i-th polynomial. The *size* of the minBase is t.

The final advice that the preprocessing algorithm produces is a minBase for every pair $(u, v) \in V \times V$ where the i-th polynomial has the property that $p_i(r)$ is the distance from u to v in $G(r)$ for each $b_i \leq r \leq b_{i+1}$.

Definition 3.2. A $minBase^{\ell}(u, v)$ is a minBase corresponding to the ordered pair u, v, where the i-th polynomial p_i has the property that for $r \in [b_i, b_{i+1}]$, $p_i(r)$ is the length of a shortest path from u to v in $G(r)$, that is taken among all paths that use at most 2^{ℓ} edges.
A $minBase^{\ell}(u, w, v)$ is a minBase corresponding to the ordered triple (u, w, v) where the i-th polynomial p_i has the property that for each $r \in [b_i, b_{i+1}]$, $p_i(r)$ is the sum of the lengths of a shortest path from u to w in $G(r)$, among all paths that use at most 2^{ℓ} edges, and a shortest path from w to v in $G(r)$, among all paths that use at most 2^{ℓ} edges.

Note that in both of the above definitions some of the polynomials can be $+\infty$ or $-\infty$.

Definition 3.3. If B_1 and B_2 are two minBases (not necessarily of the same size), with polynomials p^1_i and p^2_j, we say that another minBase with breaks b'_k and polynomials p'_k is $\min(B_1 + B_2)$ if the following holds.

(1) For all k there exist i, j such that $p'_k = p^1_i + p^2_j$, and
(2) For $b'_k \leq r \leq b'_{k+1}$ and for all i, j we have $p'_k(r) \leq p^1_i(r) + p^2_j(r)$.

Definition 3.4. If $B_1, B_2, \ldots, B_s$ are s minBases (not necessarily of the same size), with polynomials $p^1_{i_1}, p^2_{i_2}, \ldots, p^s_{i_s}$, another minBase with breaks b'_k and polynomials p'_k is $\min\{B_1, B_2, \ldots, B_s\}$ if the following holds.

(1) For all k there exist q such that $p'_k = p^q_{i_q}$, and
(2) For $b'_k \leq r \leq b'_{k+1}$ and for all $1 \leq q \leq s$ and all i_q, we have $p'_k(r) \leq p^q_{i_q}(r)$.

Note that using the above definition we can write the following two equations:

$$minBase^{\ell+1}(u, v) = \min_{w \in V} \left\{ minBase^{\ell}(u, w, v) \right\} \ . \tag{3.1}$$

$$minBase^{\ell}(u,w,v) = min\left(minBase^{\ell}(u,w) + minBase^{\ell}(w,v)\right)\ . \tag{3.2}$$

The following claim will prove the result. The proof of the claim is omitted due to lack of space.

Claim 3.5. If B_1 and B_2 are two minBases of sizes t_1 and t_2 respectively, then

(a) $\min(B_1 + B_2)$ can be computed from B_1 and B_2 in time $O(t_1 + t_2)$.

(b) $\min\{B_1, B_2\}$ can be computed from B_1 and B_2 in time $O(f(d)(t_1 + t_2))$, where $f(d)$ is the time required to compute the intersection points of two degree-d polynomials. The size of $\min\{B_1, B_2\}$ is $O(d(t_1 + t_2))$.

In order to compute $\min\{B_1, \dots, B_s\}$ one recursively computes $X = \min\{B_1, \dots, B_{s/2}\}$ and $Y = \min\{B_{s/2+1}, \dots, B_s\}$ and then takes $\min\{X, Y\}$.

If there are no negative cycles, then the advice that the instantiation algorithm needs from the preprocessing algorithm consists of $minBase^{\lceil \log n \rceil}(u,v)$. To deal with negative cycles, both $minBase^{\lceil \log n \rceil}(u,v)$ and $minBase^{\lceil \log n \rceil + 1}(u,v)$ are produced, and the instantiation algorithm compares them. if they are not equal, then the correct output is $-\infty$.

Also note that $minBase^{0}(u,v)$ is the trivial minBase where the breaks are $-\infty$ and $+\infty$ and the polynomial is weight $W((u,v))$ associated to the edge (u,v) if $(u,v) \in E$ and $+\infty$ otherwise.

If the size of $minBase^{\ell}(u,v)$ is s_ℓ, then by (3.1), (3.2), and by Claim 3.5 the time to compute $minBase^{\ell+1}(u,v)$ is $O(f(d))^{\log n} s_\ell$ and the size of $minBase^{\ell+1}(u,v)$ is $O(d)^{\log n} s_\ell$. Thus one can compute the advice for u and v in time

$$(O(f(d))^{\log n})^{\log n} = O(n^{(O(1)+\log f(d)) \log n})\ ,$$

and the length of the advice string is $O(n^{(O(1)+\log d) \log n})$.

4. Proof of Theorem 1.3

Given the linear-weighted graph $G = (V, E, W)$, our preprocessing phase begins by verifying that for all $r \in [\alpha, \beta]$, $G(r)$ has no negative weight cycles. From the proof of Lemma 2.2 we know that this holds if and only if both $G(\alpha)$ and $G(\beta)$ have no negative weight cycles. This, in turn, can be verified in $O(mn)$ time using the Bellman-Ford algorithm. We may now assume that $G(r)$ has no negative cycles for any $r \in [\alpha, \beta]$. Moreover, since our preprocessing algorithm will solve a large set of shortest path problems, each of them on a specific instantiation of G, we will first compute the reweighing functions $g_v^{[\alpha,\beta]}$ of Lemma 2.2 which will enable us to apply, in some cases, algorithms that assume nonnegative edge weights. Recall that by Lemma 2.2, the functions $g_v^{[\alpha,\beta]}$ for all $v \in V$ are computed in $O(mn)$ time.

The advice constructed by the preprocessing phase is composed of two distinct parts, which we respectively call the *crude-short* advice and the *refined-long* advice. We now describe each of them.

For each edge $e \in E$, the weight is a linear function $w_e = a_e + xb_e$. Set $K = 8(\beta - \alpha)\max_e |a_e|$. Let $N_0 = \lceil K\sqrt{n} \ln n/\epsilon \rceil$ and let $N_1 = \lceil Kn/\epsilon \rceil$. We define $N_0 + 1$ and $N_1 + 1$ points in $[\alpha, \beta]$ and solve certain variants of shortest path problems instantiated in these points.

Consider first the case of splitting $[\alpha, \beta]$ into N_0 intervals. Let $\rho_0 = (\beta - \alpha)/N_0$ and consider the points $\alpha + i\rho_0$ for $i = 0, \dots, N_0$. The crude-short part of the preprocessing algorithm solves $N_0 + 1$ *limited* all-pairs shortest path problems in $G(\alpha + i\rho_0)$ for $i = 0, \dots, N_0$. Set $t = 4\sqrt{n} \ln n$, and let $d_i(u,v)$ denote the length of a shortest path from u to v in $G(\alpha + i\rho_0)$ that is chosen among all paths containing at most t vertices (possibly $d_i(u,v) = \infty$ if no such path exists). Notice that

$d_i(u, v)$ is not necessarily the distance from u to v in $G(\alpha + i\rho_0)$, since the latter may require more than t vertices. It is straightforward to compute shortest paths limited to at most k vertices (for any $1 \leq k \leq n$) in a real-weighted directed graph with n vertices in time $O(n^3 \log k)$ time, by the repeated squaring technique. In fact, they can be computed in $O(n^3)$ time (saving the $\log k$ factor) using the method from [1], pp. 204–206. This algorithm also constructs the predecessor data structure that represents the actual paths. It follows that for each ordered pair of vertices u, v and for each $i = 0, \ldots, N_0$, we can compute $d_i(u, v)$ and a path $p_i(u, v)$ yielding $d_i(u, v)$ in $G(\alpha+i\rho_0)$ in $O(n^3|N_0|)$ time which is $O(n^{3.5} \ln n)$. We also maintain, at no additional cost, linear functions $f_i(u, v)$ which sum the linear functions of the edges of $p_i(u, v)$. Note also that if $d_i(u, v) = \infty$ then $p_i(u, v)$ and $f_i(u, v)$ are undefined.

Consider next the case of splitting $[\alpha, \beta]$ into N_1 intervals. Let $\rho_1 = (\beta - \alpha)/N_1$ and consider the points $\alpha + i\rho_1$ for $i = 0, \ldots, N_1$. However, unlike the crude-short part, the refined-long part of the preprocessing algorithm cannot afford to solve an all-pairs shortest path algorithm for each $G(\alpha + i\rho_1)$, as the overall running time will be too large. Instead, we randomly select a set $H \subset V$ of (at most) $\sqrt{n}$ vertices. H is constructed by performing $\sqrt{n}$ independent trials, where in each trial, one vertex of V is chosen to H uniformly at random (notice that since the same vertex can be selected to H more than once $|H| \leq \sqrt{n}$). For each $h \in H$ and for each $i = 0, \ldots, N_1$, we solve the single source shortest path problem in $G(\alpha + i\rho_1)$ from h, and also (by reversing the edges) solve the single-destination shortest path *toward* h. Notice that by using the reweighing functions $g_v^{[\alpha,\beta]}$ we can solve all of these single source problems using Dijkstra's algorithm. So, for all $h \in H$ and $i = 0, \ldots, N_1$ the overall running time is

$$O(|N_1||H|(m + n \log n)) = O(n^{1.5}m + n^{2.5} \log n) = O(n^{3.5}) .$$

We therefore obtain, for each $h \in H$ and for each $i = 0, \ldots, N_1$, a shortest path tree $T_i(h)$, together with distances $d_i^*(h, v)$ from h to each other vertex $v \in V$, which is the distance from h to v in $G(\alpha + i\rho_1)$. We also maintain the functions $f_i^*(h, v)$ that sum the linear equations on the path in $T_i^*(h)$ from h to v. Likewise, we obtain a "reversed" shortest path tree $S_i^*(h)$, together with distances $d_i^*(v, h)$ from each $v \in V$ to h, which is the distance from v to h in $G(\alpha+i\rho_1)$. Similarly, we maintain the functions $f_i^*(v, h)$ that sum the linear equations on the path in $S_i^*(h)$ from v to h.

Finally, for each ordered pair of vertices u, v and for each $i = 0, \ldots, N_1$ we compute a vertex $h_{u,v,i} \in H$ which attains $\min_{h \in H} d_i^*(u, h) + d_i^*(h, u)$. Notice that the time to construct the $h_{u,v,i}$ for all ordered pairs u, v and for all $i = 0, \ldots, N_1$ is $O(n^{3.5})$. This concludes the description of the preprocessing algorithm. Its overall runtime is thus $O(n^{3.5} \ln n)$.

We now describe the instantiation phase. Given $u, v \in V$ and $r \in [\alpha, \beta]$ we proceed as follows. Let i be the index for which the number of the form $\alpha + i\rho_0$ is closest to r. As we have the advice $f_i(u, v)$, we let $w_0 = f_i(u, v)(r)$ (recall that $f_i(u, v)$ is a function). Likewise, let j be the index for which the number of the form $\alpha + j\rho_1$ is closest to r. As we have the advice $h = h_{u,v,j}$, we let $w_1 = f_j^*(u, h)(r) + f_j^*(h, u)(r)$. Finally, our answer is $z = \min\{w_0, w_1\}$. Clearly, the instantiation time is $O(1)$. Notice that if we also wish to output a path of weight z in $G(r)$ we can easily do so by using either $p_i(u, v)$, in the case where $z = w_0$ or using $S_j^*(h)$ and $T_j^*(h)$ (we take the path from u to h in $S_j^*(h)$ and concatenate it with the path from h to v in $T_j^*(h)$) in the case where $z = w_1$.

It remains to show that, with very high probability, the result z that we obtain from the instantiation phase is at most ϵ larger than the distance from u to v in $G(r)$. For this purpose, we first need to prove that the random set H possesses some "hitting set" properties, with very high probability.

For every pair of vertices u and v and parameter r, let $p_{u,v,r}$ be a shortest path in $G(r)$ among all simple paths from u to v containing at least $t = 4\sqrt{n} \ln n$ vertices (if G is strongly connected then such a path always exist, and otherwise we can just put $+\infty$ for all u, v pairs for which no such path exists). The following simple lemma is used in an argument similar to one used in [18].

Lemma 4.1. *For fixed u, v and r, with probability at least $1 - o(1/n^3)$ the path $p_{u,v,r}$ contains a vertex from H.*

Proof. Indeed, the path from $p_{u,v,r}$ by its definition has at least $4\sqrt{n}\ln n$ vertices. The probability that all of the $\sqrt{n}$ independent selections to H failed to choose a vertex from this path is therefore at most

$$\left(1 - \frac{4\sqrt{n}\ln n}{n}\right)^{\sqrt{n}} < e^{-4\ln n} < \frac{1}{n^4} = o(1/n^3)\ .$$

■

Let us return to the proof of Theorem 1.3. Suppose that the distance from u to v in $G(r)$ is δ. We will prove that with probability $1 - o(1)$, H is such that for every u, v and r we have $z \le \delta + \epsilon$ (clearly $z \ge \delta$ as it is the precise length of some path in $G(r)$ from u to v). Assume first that there is a path p of length δ in $G(r)$ that uses less than $4\sqrt{n}\ln n$ edges. Consider the length of p in $G(\alpha + i\rho_0)$. When going from r to $\alpha + i\rho_0$, each edge e with weight $a_e x + b_e$ changed its length by at most $|a_e|\rho_0$. By the definition of K, this is at most $\rho_0 K/(8(\beta - \alpha))$. Thus, p changed its weight by at most

$$(4\sqrt{n}\ln n) \cdot \rho_0 \frac{K}{8(\beta - \alpha)} = (4\sqrt{n}\ln n)\frac{K}{8N_0} < \frac{\epsilon}{2}.$$

It follows that the length of p in $G(\alpha + i\rho_0)$ is less than $\delta + \epsilon/2$. But $p_i(u, v)$ is a shortest path from u to v in $G(\alpha + i\rho_0)$ of all the paths that contain at most t vertices. In particular, $d_i(u, v) \le \delta + \epsilon/2$. Consider the length of $p_i(u, v)$ in $G(r)$. The same argument shows that the length of $p_i(u, v)$ in $G(r)$ changed by at most $\epsilon/2$. But $w_0 = f_i(u, v)(r)$ is that weight, and hence $w_0 \le \delta + \epsilon$. In particular, $z \le \delta + \epsilon$.

Assume next that every path of length δ in $G(r)$ uses at least $4\sqrt{n}\ln n$ edges. Let p be one such path. When going from r to $r' = \alpha + j\rho_1$, each edge e with weight $a_e x + b_e$ changed its length by at most $|a_e|\rho_1$. By the definition of K, this is at most $\rho_1 K/(8(\beta - \alpha))$. Thus, p changed its weight by at most

$$n \cdot \rho_1 \frac{K}{8(\beta - \alpha)} = n\frac{K}{8N_1} < \frac{\epsilon}{8}.$$

In particular, the length of $p_{u,v,r'}$ is not more than the length of p in $G(r')$, which, in turn, is at most $\delta + \epsilon/8$. By Lemma 4.1, with probability $1 - o(1/n^3)$, some vertex of h appears on $p_{u,v,r'}$. Moreover, by the union bound, with probability $1 - o(1)$ *all* paths of the type $p_{u,v,r'}$ (remember that r' can hold one of $O(n)$ possible values) are thus covered by the set H. Let h' be a vertex of H appearing in $p_{u,v,r'}$. We therefore have $d^*_j(u, h') + d^*_j(h', v) \le \delta + \epsilon/8$. Since $h = h_{u,v,j}$ is taken as the vertex which minimizes these sums, we have, in particular, $d^*_j(u, h) + d^*_j(h, v) \le \delta + \epsilon/8$. Consider the path q in $G(\alpha + j\rho_1)$ realizing $d^*_j(u, h) + d^*_j(h, v)$. The same argument shows that the length of q in $G(r)$ changed by at most $\epsilon/8$. But $w_1 = f^*_j(u, h)(r) + f^*_j(h, v)(r)$ is that weight, and hence $w_1 \le \delta + \epsilon/4$. In particular, $z \le \delta + \epsilon/4$.

5. Concluding remarks

We have constructed several parametric shortest path algorithms, whose common feature is that they preprocess the generic instance and produce an advice that enables particular instantiations to be solved faster than running the standard weighted distance algorithm from scratch. It would be of interest to improve upon any of these algorithms, either in their preprocessing time or in their instantiation time, or both.

Perhaps the most challenging open problem is to improve the preprocessing time of Theorem 1.2 to a polynomial one, or, alternatively, prove an hardness result for this task. Perhaps less ambitious is the preprocessing time in Theorem 1.1.

Finally, parametric algorithms are of practical importance for other combinatorial optimization problems as well. It would be interesting to find applications where, indeed, a parametric algorithm can be truly beneficial, as it is in the case of shortest path problems.

Acknowledgment

We thank Oren Weimann and Shay Mozes for useful comments.

References

[1] A. V. Aho, J. E. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Longman Publishing Co., Boston, MA, 1974.

[2] R. Bellman, *On a routing problem*, Quarterly of Applied Mathematics 16 (1958), 87–90.

[3] P. Carstensen, *The complexity of some problems in parametric linear and combinatorial programming*, Ph.D. Thesis, Mathematics Dept., U. of Michigan, Ann Arbor, Mich., 1983.

[4] T. M. Chan, *More Algorithms for All-Pairs Shortest Paths in Weighted Graphs*, Proceedings of the 39^{th} ACM Symposium on Theory of Computing (STOC), ACM Press (2007), 590–598.

[5] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation 9 (1990), 251–280.

[6] E. W. Dijkstra, *A note on two problems in connection with graphs*, Numerische Mathematik 1 (1959), 269–271.

[7] R. W. Floyd, *Algorithm 97: shortest path* Communications of the ACM 5 (1962), 345.

[8] M. L. Fredman, *New bounds on the complexity of the shortest path problem*, SIAM Journal on Computing 5 (1976), 49–60.

[9] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, Journal of the ACM 34 (1987), 596–615.

[10] D. Gusfield *Parametric combinatorial computing and a problem of program module distribution*, Journal of the ACM 30(3) (1983), 551–563.

[11] C. P. M. van Hoesel, A. W. J. Kolen, A. H. G. Rinooy and A. P. M. Wagelmans, *Sensitivity analysis in combinatorial optimization: a bibliography*. Report 8944/A, Econometric Institute, Erasmus University Rotterdam, (1989).

[12] D. B. Johnson, *Efficient algorithms for shortest paths in sparse graphs*, Journal of the ACM 24 (1977), 1–13.

[13] N. Megiddo, *Combinatorial Optimization with Rational Objective Functions*, Mathematics of Operation Research Vol.4 No.4 (1979), 414–424.

[14] K. Murty. *Computational complexity of parametric linear programming*. Math. Programming-19, (1980) 213–219.

[15] R. M. Karp and J. B. Orlin, *Parametric shortest path algorithms for with an application to cycle staffing*, Discrete Applied Mathematics 3 (1981), 37–45.

[16] E. Nikolova, J. A. Kelner, M. Brand and M. Mitzenmacher, *Stochastic Shortest Paths Via Quasi-convex Maximization*, Proceedings of the 14^{th} Annual European Symposium on Algorithms (ESA), LNCS (2006), 552–563.

[17] N. E. Young, R. E. Tarjan and J. B. Orlin, *Faster parametric shortest path and minimum-balance algorithms*, Networks 21 (1991), 205–221.

[18] U. Zwick, *All-pairs shortest paths using bridging sets and rectangular matrix multiplication*, Journal of the ACM 49 (2002), 289–317.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 179-190
www.stacs-conf.org

CONTINUOUS MONITORING OF DISTRIBUTED DATA STREAMS OVER A TIME-BASED SLIDING WINDOW

HO-LEUNG CHAN[1] AND TAK-WAH LAM[1] AND LAP-KEI LEE[2] AND HING-FUNG TING[1]

[1] Department of Computer Science, University of Hong Kong, Hong Kong
E-mail address: {hlchan,twlam,hfting}@cs.hku.hk

[2] Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany
E-mail address: lklee@mpi-inf.mpg.de

Abstract. The past decade has witnessed many interesting algorithms for maintaining statistics over a data stream. This paper initiates a theoretical study of algorithms for monitoring distributed data streams over a time-based sliding window (which contains a variable number of items and possibly out-of-order items). The concern is how to minimize the communication between individual streams and the root, while allowing the root, at any time, to be able to report the global statistics of all streams within a given error bound. This paper presents communication-efficient algorithms for three classical statistics, namely, basic counting, frequent items and quantiles. The worst-case communication cost over a window is $O(\frac{k}{\varepsilon}\log\frac{\varepsilon N}{k})$ bits for basic counting and $O(\frac{k}{\varepsilon}\log\frac{N}{k})$ words for the remainings, where k is the number of distributed data streams, N is the total number of items in the streams that arrive or expire in the window, and $\varepsilon < 1$ is the desired error bound. Matching and nearly matching lower bounds are also obtained.

1. Introduction

The problems studied in this paper are best illustrated by the following puzzle. John and Mary work in different laboratories and communicate by telephone only. In a forever-running experiment, John records which devices have an exceptional signal in every 10 seconds. To adjust her devices, Mary at any time needs to keep track of the number of exceptional signals generated by each device of John in the last one hour. John can call Mary every 10 seconds to report the exceptional signals, yet this requires too many calls in an hour and the total message size per hour is linear to the total number N of exceptional signals in an hour. Mary's devices actually allow some small error. Can the number of calls and message size be reduced to $o(N)$, or even poly-log N if a small error (say, 0.1%) is

1998 ACM Subject Classification: F.2.2 [Analysis of algorithms and problem complexity]: Nonnumerical algorithms and problems.

Key words and phrases: Algorithms, distributed data streams, communication efficiency, frequent items.

T.W. Lam is partially supported by the GRF Grant HKU-713909E; H.F. Ting is partially supported by the GRF Grant HKU-716307E.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2453

allowed? It is important to note that the input is given online and Mary needs to know the answers continuously; this makes our problem different from those in other similar classical models, such as the Simultaneous Communication Complexity model [4], in which all inputs are given in advance and the parties need to compute an answer only once.

Motivation. The above problem appears in data stream applications, e.g., network monitoring or stock analysis. In the last decade, algorithms for continuous monitoring of a single massive data stream gained a lot of attention (see [1, 26] for a survey), and the main challenge has been how to represent the massive data using limited space, while allowing certain statistics (e.g., item counts, quantiles) to be computed with sufficient accuracy.

The space-accuracy tradeoff for representing a single stream has gradually been understood over the years (e.g., [2, 15, 18, 19]). Recently, motivated by large scale networks, the database community is enthusiastic about communication-efficient algorithms for continuous monitoring of multiple, distributed data streams. In such applications, we have $k \geq 1$ remote sites each monitoring a data stream, and there is a root (or coordinator) responsible for computing some global statistics. A remote site needs to maintain certain statistics itself, and has to communicate with the root often enough so that the root can compute, at any time, the statistics of the union of all data streams within a certain error. The objective is to minimize the communication. The communication aspects of data streams introduce several challenging theoretical questions such as what is the optimal communication-accuracy tradeoff for maintaining a particular statistic, and whether two-way communication is inherently more efficient than one-way communication.

Data stream models and ε-approximate queries. The data stream at each remote site is a sequence of items from a totally ordered set U. Each item is associated with an integral time-stamp recording its arrival time. Each remote site has limited space and hence it can only maintain the required statistics approximately. The statistics can be based on the whole data stream [2, 15, 18, 19] or only the recent items [3, 14, 22]. Recent items can be modeled by two types of sliding windows [5, 13]. Let W be the window size, which is a positive integer. The *count-based sliding window* includes the last W items in the data stream, while the *time-based sliding window* includes items whose time-stamps are within the last W time units. The latter assumes that zero or more items can arrive at a time. Items in a sliding window will expire and are more difficult to handle than in the whole data stream. For example, counting the frequency of a certain item in the whole stream can be done easily by maintaining a single counter, yet the same problem requires space $\Theta(\frac{1}{\varepsilon}\log^2(\varepsilon W))$ bits for a count-based sliding window even if we allow a relative error of at most ε [13, 16]. In fact, the whole data stream model can be viewed as a special case of the sliding window model with window size being infinite. Also, a count-based window is a special case of a time-based window in which exactly one item arrives at a time. This paper focuses on time-based window, and the algorithms are applicable to the other two models.

We study algorithms that enable the root to answer three types of classical ε-approximate queries, defined as follows. Let $0 < \varepsilon < 1$. For any stream σ, let $c_{j,\sigma}$ and c_σ be the count of item j and all items whose timestamps are in the current window, respectively. Denote $c_j = \sum_\sigma c_{j,\sigma}$ and $c = \sum_\sigma c_\sigma$ as the total count of item j and all items in all the data streams, respectively.

- *Basic Counting.* Return an estimate $\hat{c}$ on the total count c such that $|\hat{c} - c| \leq \varepsilon c$. (Note that this query can be generalized to count data items of a fixed subset $X \subseteq U$; the literature often refers to the special case with $U = \{0, 1\}$ and $X = \{1\}$.)

- *Frequent Items.* Given any $0 < \phi < 1$, return a set $F \subseteq U$ which includes all items j with $c_j \geq \phi c$ and possibly some items j' with $c_{j'} \geq \phi c - \varepsilon c$.
- *Quantiles.* Given any $0 < \phi < 1$, return an item whose rank is in $\left[\phi c - \varepsilon c,\ \phi c + \varepsilon c\right]$ among the c items in the current sliding window.

As in most previous works, we need to answer the following type of ε-approximate queries in order to answer queries on frequent items.

- *Approximate Counting.* Given any item j, return an estimate $\hat{c}_j$ such that $|\hat{c}_j - c_j| \leq \varepsilon c$. (Note that this query gives estimate for any item, not just the frequent items. Also, the error bound is in term of c, which may be much larger than c_j.)

We need an algorithm to determine when and how the remote sites communicate with the root so that the root can answer the queries at any time. The objective is to minimize the worst-case communication cost within a window of W time units.

Previous works. Recently, the database literature has a flurry of results on continuous monitoring of distributed data streams, e.g. [6, 8, 9, 12, 17, 20, 24, 25, 27, 28]. The algorithms studied can be classified into two types: *one-way* algorithms only allow messages sent from each remote site to the root, and *two-way* algorithms allow bi-directional communication between the root and each site. One-way algorithms are often very simple as a remote site has little information and all it can do is to update the root when its local statistics deviate significantly from those previously sent. On the other hand, most two-way algorithms are complicated and often involve non-trivial heuristics. It is commonly believed in the database community that two-way algorithms are more efficient; however, for most existing two-way algorithms, their worst-case communication costs are still waiting for rigorous mathematical analysis, and existing works often rely on experimental results when evaluating the communication cost.

The literature contains several results on the mathematical analysis of the worst-case performance of one-way algorithms. They are all for the whole data stream setting. Keralapura et al. [21] studied the thresholded-count problem, which leads to an algorithm for basic counting with communication cost $O(\frac{k}{\varepsilon} \log \frac{N}{k})$ words, where k and N are the number of streams and the number of items in these streams, respectively. Cormode et al. [9] gave an algorithm for quantiles with communication cost $O(\frac{k}{\varepsilon^2} \log \frac{N}{k})$ words per stream. They also showed how to handle frequent items via a reduction to quantiles, so the communication cost remains the same. More recently, Yi and Zhang [29] have reduced the communication cost for frequent items to $O(\frac{k}{\varepsilon} \log \frac{N}{k})$ words, and quantile to $O(\frac{k}{\varepsilon} \log^2(\frac{1}{\varepsilon}) \log \frac{N}{k})$ words, using some two-way algorithms; these are the only analyses for two-way algorithms so far.

There have been attempts to devise heuristics to extend some whole-data-stream algorithms to sliding windows, yet not much has been known about their worst-case performance. For example, Cormode et al. [9] have extended their algorithms for quantiles and frequent items to sliding windows. They believed that the communication cost would only have a mild increase, but no supporting analysis has been given. The analysis of sliding-window algorithms is more difficult because the expiry of items destroys some monotonic property that is important to the analysis for whole data stream. In fact, finding sliding-window algorithms with efficient worst-case communication has been posed as an open problem in the latest work of Yi and Zhang [29].

Our results. This paper gives the first mathematical analysis of the communication cost in the sliding window model. We derive lower bounds on the worst-case communication cost of any two-way algorithm (and hence any one-way algorithm) for answering the four

	Basic Counting (**bits**)	Approximate Counting/ Frequent items (**words**)	Quantiles (**words**)
Whole data stream	$O(\frac{k}{\varepsilon}\log\frac{N}{k})$ words [21] $\Theta(\frac{k}{\varepsilon}\log\frac{\varepsilon N}{k})$ bits	$O(\frac{k}{\varepsilon}\log\frac{N}{k})$ [29]	$O(\frac{k}{\varepsilon}\log^2(\frac{1}{\varepsilon})\log\frac{N}{k})$ [29]
		$\Omega(\frac{k}{\varepsilon}\log\frac{\varepsilon N}{k})$ [29, 30]	
Sliding window	$\Theta(\frac{k}{\varepsilon}\log\frac{\varepsilon N}{k})$	$O(\frac{k}{\varepsilon}\log\frac{N}{k})$	$O(\frac{k}{\varepsilon^2}\log\frac{N}{k})$
		$\Omega(\frac{k}{\varepsilon}\log\frac{\varepsilon N}{k})$	
Sliding window & out-of-order	$O((\frac{W}{W-\tau})\frac{k}{\varepsilon}\log\frac{\varepsilon N}{k})$	$O((\frac{W}{W-\tau})\frac{k}{\varepsilon}\log\frac{N}{k})$	$O((\frac{W}{W-\tau})\frac{k}{\varepsilon^2}\log\frac{N}{k})$
	$\Omega(\max\{\frac{W}{W-\tau}, \frac{k}{\varepsilon}\log\frac{\varepsilon N}{k}\})$	$\Omega(\max\{\frac{W}{W-\tau}, \frac{k}{\varepsilon}\log\frac{\varepsilon N}{k}\})$	

Table 1: Bounds on the communication costs. Note that the bounds are stated in bits for basic counting, and in words for the other problems.

types of ε-approximate queries. These lower bounds hold even when each remote site has unlimited space to maintain the local statistics exactly. More interestingly, we analyze some common-sense algorithms that use one-way communication only and prove that their communication costs match or nearly match the corresponding lower bounds. In our algorithms, each remote site only needs to maintain some $\Theta(\varepsilon)$-approximate statistics for its local data, which actually adds more complication to the problem. These results demonstrate optimal or near optimal communication-accuracy tradeoffs for supporting these queries over the sliding window. Our work reveals that two-way algorithms could not be much better than one-way algorithms in the worst case.

Below we state the lower and upper bounds precisely. Recall that there are k remote sites and the sliding window contains W time units. We prove that within any window, the root and the remote sites need to communicate, in the worst case, $\Omega(\frac{k}{\varepsilon}\log\frac{\varepsilon N}{k})$ bits for basic counting and $\Omega(\frac{k}{\varepsilon}\log\frac{\varepsilon N}{k})$ words for the other three queries, where N is the total number of items arriving or expiring within that window.[1] For upper bounds, our analysis shows that basic counting requires $O(\frac{k}{\varepsilon}\log\frac{\varepsilon N}{k})$ bits within any window, and approximate counting $O(\frac{k}{\varepsilon}\log\frac{N}{k})$ words. The estimates given by approximate counting are sufficient to find frequent items, hence the latter problem has the same communication cost. For quantiles, it takes $O(\frac{k}{\varepsilon^2}\log\frac{N}{k})$ words. See the second row (sliding window) of Table 1 for a summary.

As mentioned before, sliding-window algorithms can be applied to handle the special case of whole data streams in which the window size W is infinite and N is the total number of arrived items. The first row of Table 1 shows the results on whole data streams. Our work has improved the communication cost for basic counting from $O(\frac{k}{\varepsilon}\log\frac{N}{k})$ words [21] to $O(\frac{k}{\varepsilon}\log\frac{\varepsilon N}{k})$ bits. For approximate counting and frequent items, our work implies a one-way algorithm with communication cost of $O(\frac{k}{\varepsilon}\log\frac{N}{k})$ words; this matches the performance of the two-way algorithm by Yi and Zhang [29]. In their algorithm, the root regularly updates every remote site about the global count of all items. In contrast, we use the idea that

[1] Note that the number of items arriving or expiring within window $[t-W+1, t]$ is no greater than the number of items arriving within $[t-2W+1, t]$.

items with small count could be "turned off" for further updating. As a remark, our upper bound on quantiles is $O(\frac{k}{\varepsilon^2}\log\frac{N}{k})$ words which is weaker than that of [29].

Our algorithms can be readily applied to out-of-order streams [7, 10]. In an out-of-order stream, each item is associated with an integral time-stamp recording its creation time, which may be different from its arrival time. We say that the stream has *tardiness* τ if any item with time-stamp t must arrive within τ time units from t, i.e., at any time in $[t, t+\tau]$. Without loss of generality, we assume that $\tau \in \{0, 1, 2, \ldots, W-1\}$ (if an item time-stamped at t arrives after $t + W - 1$, it has already expired and can be ignored). Note that for any data stream with tardiness greater than zero, the items may not be arriving in non-decreasing order of their time-stamps. Our previous discussion of data streams assumes tardiness equal to 0, and such data streams are called *in-order* data streams. The previous lower bounds for in-order streams are all valid in the out-of-order setting. In addition, we obtain lower bounds related to τ, namely, $\Omega(\frac{W}{W-\tau})$ bits for basic counting and $\Omega(\frac{W}{W-\tau})$ words for the other three problems. Regarding upper bounds, our algorithms when applied to out-of-order streams with tardiness τ will just increase the communication cost by a factor of $\frac{W}{W-\tau}$. The results are summarized in the last row of Table 1.

The idea for basic counting is relatively simple. As the root does not require an exact total count, each data stream can communicate to the root only when its local count increases or decreases by a certain ratio $\varepsilon > 0$; we call such a communication step an *up* or *down* event, respectively. To answer the total count of all streams, the root simply sums up all the individual counts it has received. It is easy to prove that this answer is within some desired error bound. If each count is over the whole stream (i.e., window size $= \infty$ and N is the total number of arrived items), the count is increasing and there is no down event. A stream would have at most $O(\log_{1+\varepsilon} N)$ up events and the communication cost is at most that many words. However, the analysis becomes non-trivial in a sliding time window. Now items can expire and down events can occur. An up event may be followed by some down events and the count is no longer increasing. The tricky part is to find a new measure of progress. We identify a "characteristic set" of each up event such that each up event must increase the size of this set by a factor of at least $1+\varepsilon$, hence bounding the number of up events to be $O(\log_{1+\varepsilon} N)$. Down events are bounded using another characteristic set. Due to space limitation, the details can only be given in the full paper.

Approximate counting of all possible items is much more complicated, which will be covered in details in the rest of this paper. Assuming in-order streams, we derive and analyze two algorithms for approximate counting in Section 2. In Section 3, we discuss frequent items, quantiles, and finally out-of-order streams. The lower bound results are relatively simple and omitted due to space limitation.

2. Approximate Counting of all items

This section presents algorithms for the streams to communicate to the root so that the root at any time can approximate the count of each item. As a warm-up, we first consider the simple algorithm in which a stream will inform the root whenever its count of an item increases or decreases by a certain fraction of its total item count. We show in Section 2.1 that within any window of W time units, each data stream σ_i $(1 \le i \le k)$ needs to send at most $O((\Delta + \frac{1}{\varepsilon})\log n_i)$ words to the root, where Δ is the number of distinct items and n_i is the number of items of σ_i that arrive or expire within the window. Then, the total communication cost within this window is $\sum_{1\le i\le k}(\Delta + \frac{1}{\varepsilon})\log n_i$, which, by

Jensen's inequality, is no greater than $(\Delta + \frac{1}{\varepsilon})k \log(\sum_{1\le i\le k} n_i)/k = (\Delta + \frac{1}{\varepsilon})k \log \frac{N}{k}$ where $N = \sum_{1\le i\le k} n_i$. We then modify the algorithm so that a stream can "turn off" items whose counts are too small, and we give a more complicated analysis to deal with the case when many such items increase their counts rapidly (Section 2.2). The communication cost is reduced to $O(\frac{k}{\varepsilon} \log \frac{N}{k})$ words, independent of Δ.

2.1. A simple algorithm

Consider any stream σ. At any time t, let $c(t)$ and $c_j(t)$ be the number of all items and item j arriving at σ in $[t - W + 1, t]$, respectively. Let $\lambda < 1/11$ be a positive constant (which will be set to $\varepsilon/11$). We maintain two λ-approximate data structures [13, 23] at σ locally, which can report estimates $\hat{c}(t)$ and $\hat{c}_j(t)$ for $c(t)$ and $c_j(t)$, respectively, such that [2]

$$(1-\lambda/6)c(t) \le \hat{c}(t) \le (1+\lambda/6)c(t); \quad \text{and} \quad c_j(t) - \lambda c(t) \le \hat{c}_j(t) \le c_j(t) + \lambda c(t).$$

Simple algorithm. At any time t, for any item j, let $p < t$ be the last time $\hat{c}_j(p)$ is sent to the root. The stream sends the estimate $\langle j, \hat{c}_j(t)\rangle$ to the root if the following event occurs.

- *Up:* $\hat{c}_j(t) > \hat{c}_j(p) + 9\lambda\hat{c}(t)$.
- *Down:* $\hat{c}_j(t) < \hat{c}_j(p) - 9\lambda\hat{c}(t)$.

Root's perspective. At any time t, let $r_{j,\sigma}(t)$ be the last estimate received from a stream σ for item j (at or before t). The root can estimate the total count of item j over all streams by summing all $r_{j,\sigma}(t)$ received. More precisely, for any $0 < \varepsilon < 1$, we set $\lambda = \varepsilon/11$ and let each stream use the simple algorithm. Then for each stream σ, the approximate data structures for $\hat{c}_j(t)$ and $\hat{c}(t)$ together with the simple algorithm guarantee that $c_j(t) - 11\lambda c(t) \le r_{j,\sigma}(t) \le c_j(t) + 11\lambda c(t)$. Summing $r_{j,\sigma}(t)$ over all streams would give the root an estimate of the total count of item j within an error of ε of the total count of all items.

Communication Complexity. At any time t, we denote the reference window as $[t_o, t]$, where $t_o = t - W + 1$. Let n be the number of items of σ that arrive or expire in $[t_o, t]$. Assume that there are at most Δ distinct items. We first show that a stream σ encounters $O((\frac{1}{\lambda} + \Delta)\log n)$ up events and sends $O((\frac{1}{\lambda} + \Delta)\log n)$ words within $[t_o, t]$. The analysis of down events is similar and will be detailed later. For any time $t_1 \le t_2$, it is useful to define $\sigma_{[t_1,t_2]}$ (resp. $\sigma_{j,[t_1,t_2]}$) as the multi-set of all items (resp. item j only) arriving at σ within $[t_1, t_2]$, and $|\sigma_{[t_1,t_2]}|$ as the size of this multi-set.

Consider an up event U_j of some item j that occurs at time $v \in [t_o, t]$. Define the *previous event* of U_j to be the latest event (up or down) of item j that occurs at time $p < v$. We call p the *previous-event time* of U_j. The number of up events with previous-event time before t_o is at most Δ. To upper bound the number of up events with previous-event time $p \ge t_o$ is, however, non-trivial; below we call such an up event a *follow-up* (event). Intuitively, a follow-up can be triggered by frequent arrivals of an item, or mainly the relative decrease of the total count. This motivates us to classify follow-ups into two types and analyze them differently. A follow-up U_j is said to be *absolute* if $c(p) \le \frac{6}{5}c(v)$, and *relative* otherwise. Define *Recent-items*(U_j) to be the multi-set of item j's that arrive after the previous event of U_j, i.e., *Recent-items*$(U_j) = \sigma_{j,[p+1,v]}$.

[2] The constant 6 in the inequality is arbitrary. It can be replaced with any number provided that other constants in the algorithm and analysis (e.g., the constant 9 in definition of up events) are adjusted accordingly.

Absolute follow-ups. To obtain a tight bound of absolute follow-ups, we need a characteristic-set argument that can consider the growth of different items together. Let $t_1, t_2, ..., t_k$ be the times in $[t_o, t]$ when some absolute follow-ups (of one or more items) occur. Let x_i be the number of items having an absolute follow-up at t_i. Note that for all i, $x_i \leq \min\{1/(7\lambda), \Delta\}$,[3] and $\sum_{i=1}^{k} x_i$ is the number of absolute follow-ups in $[t_o, t]$. We define the characteristic set S_i at each t_i as follows:

$S_i =$ the union of *Recent-items*(U_j) over all absolute follow-ups U_j occurring at $t_1, t_2, \ldots, t_i$.

Recall that n is the number of items of σ that arrive or expire in $[t - W + 1, t]$.

Lemma 2.1. **(i)** *For any* $2 \leq i \leq k$, $|S_i| > (1 + 6x_i\lambda)|S_{i-1}|$. **(ii)** *There are* $\sum_{i=1}^{k} x_i = O(\frac{1}{\lambda}\log n)$ *absolute follow-ups within* $[t_o, t]$.

Proof. For (i), consider an absolute follow-up U_j of an item j, occurring at time t_i with previous-event time p_i. Note that the increase in the count of item j from p_i to t_i must be due to the recent items. We have

$$\begin{aligned}
|\textit{Recent-items}(U_j)| &\geq c_j(t_i) - c_j(p_i) \\
&\geq \hat{c}_j(t_i) - \hat{c}_j(p_i) - \lambda c(t_i) - \lambda c(p_i) && \text{(by } \sigma\text{'s local data structures)} \\
&> 9\lambda\hat{c}(t_i) - \lambda c(t_i) - \lambda c(p_i) && \text{(definition of an up event)} \\
&\geq (9\lambda(1 - \tfrac{\lambda}{6}) - \lambda - \tfrac{6}{5}\lambda)c(t_i) \geq 6\lambda c(t_i) && (U_j \text{ is absolute})
\end{aligned}$$

There are x_i absolute follow-ups at t_i, so $|S_i| > |S_{i-1}| + x_i\,(6\lambda c(t_i))$. Since $S_i \subseteq \sigma_{[t_o, t_i]}$, $c(t_i) \geq |S_i| \geq |S_{i-1}|$. Therefore, we have $|S_i| > |S_{i-1}| + 6x_i\lambda|S_i| \geq (1 + 6x_i\lambda)|S_{i-1}|$.

For (ii), we note that $n \geq |S_k| > \prod_{i=2}^{k}(1 + 6x_i\lambda)|S_1|$, and $|S_1| \geq 1$. Thus, $\prod_{i=2}^{k}(1 + 6x_i\lambda) < n$, or equivalently, $\ln n > \sum_{i=2}^{k} \ln(1 + 6x_i\lambda)$. The latter is at least $\sum_{i=2}^{k} \frac{6x_i\lambda}{1+6x_i\lambda} \geq \lambda \sum_{i=2}^{k} x_i$. The last inequality follows from that $x_i \leq 1/(7\lambda)$ for all i. Thus, $\sum_{i=1}^{k} x_i \leq x_1 + \frac{1}{\lambda}\ln n = O(\frac{1}{\lambda}\log n)$. ∎

Relative follow-ups. A relative follow-up occurs only when a lot of items expire, and relative follow-ups of the same item cannot occur too frequently. Below we define $O(\log n)$ time intervals and argue that no item can have two relative follow-ups within an interval. For an item with time-stamp t_1, we define the *first expiry time* to be $t_1 + W$. At any time u in $[t_o, t]$, define H_u to be the set of all items whose first expiry time is within $[u + 1, t]$, i.e., $H_u = \sigma_{[u-W+1, t_o-1]}$. $|H_u|$ is non-increasing as u increases. Consider the times $t_o = u_0 < u_1 < u_2 < \cdots < u_\ell \leq t$ such that for $i \geq 1$, u_i is the first time such that $|H_{u_i}| < \frac{5}{6}|H_{u_{i-1}}|$. For convenience, let $u_{\ell+1} = t + 1$. Note that $|H_{u_0}| \leq n$ and $\ell = O(\log n)$.

Lemma 2.2. **(i)** *Every item* j *has at most one relative follow-up* U_j *within each interval* $[u_i, u_{i+1} - 1]$. **(ii)** *There are at most* $O(\Delta \log n)$ *relative follow-ups within* $[t_o, t]$.

Proof. For (i), assume U_j occurs at time v in $[u_i, u_{i+1} - 1]$, and its previous event occurs at time p. By definition, $c(p) > \frac{6}{5}c(v)$. Thus,

$$|H_p| - |H_v| = |\sigma_{[p-W+1, v-W]}| \geq c(p) - c(v) > \tfrac{1}{5}c(v) \geq \tfrac{1}{5}|\sigma_{[v-W+1, t_o-1]}| = \tfrac{1}{5}|H_v| \ ,$$

and $|H_v| < \frac{5}{6}|H_p|$. Since $v < u_{i+1}$ and $|H_v| \geq \frac{5}{6}|H_{u_i}|$, we have $|H_p| > |H_{u_i}|$ and $p < u_i$. For (ii), there are Δ distinct items, so there are at most Δ relative follow-ups within each interval $[u_i, u_{i+1} - 1]$, and at most $O(\Delta \log n)$ relative follow-ups within $[t_o, t]$. ∎

[3] If an up event of an item j occurs at time t_i, then $c_j(t_i) \geq \hat{c}_j(t_i) - \lambda c(t_i) > 9\lambda\hat{c}(t_i) - \lambda c(t_i) \geq 7\lambda c(t_i)$. Thus the number of up events at time t_i is at most $c(t_i)/(7\lambda c(t_i)) = 1/(7\lambda)$.

Down events. The analysis is symmetric to that of up events. The only non-trivial thing is the definition of the characteristic set for bounding the absolute follow-downs D_j, which is defined in an opposite sense: Assume D_j occurs at time v and its previous event occurs at $p \geq t_o$. D_j is said to be *absolute* if $c(p) \leq \frac{6}{5}c(v)$. Let $\mathit{Expire}(D_j)$ be the multi-set of item j's whose first expiry time is within $[p+1, v]$. I.e., $\mathit{Expire}(D_j) = \sigma_{j,[p-W+1,v-W]}$.

It is perhaps a bit tricky that instead of defining the characteristic set of absolute follow-downs at the time they occur, we consider the times of the corresponding *previous events* of these follow-downs. Let $p_1, p_2, ..., p_k$ be the times in $[t_o, t]$ such that there is at least one event E_j (up or down) at p_i which is the previous event of an absolute follow-down D_j occurring after p_i. Let y_i be the number of such previous events at p_i, and let $AD(p_i)$ be the set of corresponding absolute follow-downs. Note that y_i (unlike x_i) only admits a trivial upper bound of Δ. We define the characteristic set T_i for each p_i as follows:

$$T_i = \text{the union of } \mathit{Expire}(D_j) \text{ over all } D_j \in AD(p_i), AD(p_{i+1}), \ldots, AD(p_k).$$

Similar to Lemma 2.1, we can show that $|T_i| > (1 + 5y_i\lambda)|T_{i+1}|$. Owing to a weaker bound of individual y_i, the number of absolute follow-downs, which equals $\sum_{i=1}^{k} y_i$, is shown to be $O((\frac{1}{\lambda} + \Delta) \log n)$.

Combining the analyses on up and down events, and let $\lambda = \varepsilon/11$, we have the following.

Theorem 2.3. *The simple algorithm sends at most $O((\frac{1}{\varepsilon} + \Delta) \log n)$ words to the root during window $[t - W + 1, t]$.*

2.2. The full algorithm

In this section, we extend the previous algorithm and give a new characteristic-set analysis that is based on future events (instead of the past events) to show that each stream's communication cost per window can be reduced to $O(\frac{1}{\varepsilon} \log n)$ words. Then, by Jensen's inequality again, we conclude that the total communication cost per window is $O(\frac{k}{\varepsilon} \log \frac{N}{k})$. Intuitively, when the estimate $\hat{c}_j(t)$ of an item j is too small, say, less than $3\lambda\hat{c}(t)$, the algorithm treats this estimate as 0 and set the off_j flag of j to be true. This restricts the number of items with a positive estimate to $O(\frac{1}{\lambda})$. Initially, the off_j flag is true for all items j. Given $0 < \lambda < \varepsilon/11$, the stream communicates with the root as follows.

Algorithm AC. At any time t, for any item j, let $p < t$ be the time the last estimate of j, i.e., $\hat{c}_j(p)$, is sent to the root. The stream sends the estimate of j to the root if the following event occurs.

- *Up:* If $\hat{c}_j(t) > \hat{c}_j(p) + 9\lambda\hat{c}(t)$, send $\langle j, \hat{c}_j(t) \rangle$ and set $\mathit{off}_j = \mathit{false}$.
- *Off:* If $\mathit{off}_j = \mathit{false}$ and $\hat{c}_j(t) < 3\lambda\hat{c}(t)$, reset $\hat{c}_j(t)$ to 0, send $\langle j, \hat{c}_j(t) \rangle$ and set $\mathit{off}_j = \mathit{true}$.
- *Down:* If $\mathit{off}_j = \mathit{false}$ and $\hat{c}_j(t) < \hat{c}_j(p) - 9\lambda\hat{c}(t)$, send $\langle j, \hat{c}_j(t) \rangle$.

It is straightforward to check that the root can answer the approximate counting query for any item. We analyze the communication complexity of different events as follows.

Fact 1. At any time v, the number of items j with $\mathit{off}_j = \mathit{false}$ is at most $\frac{1}{\lambda}$.[4]

[4] For any item j, if $\mathit{off}_j = \mathit{false}$, then $\hat{c}_j(v) \geq 3\lambda\hat{c}(v)$ and $c_j(v) \geq \hat{c}_j(v) - \lambda c(v) \geq (3\lambda(1-\lambda) - \lambda)c(v) \geq \lambda c(v)$. Thus the number of items j with $\mathit{off}_j = \mathit{false}$ is at most $c(v)/\lambda c(v) = \frac{1}{\lambda}$.

Off events. Recall that we are considering the window $[t_o, t]$, and n is the number of items arriving or expiring within $[t_o, t]$. By Fact 1, just before t_o, there are at most $\frac{1}{\lambda}$ items with $\mathit{off}_j = \mathit{false}$. Within $[t_o, t]$, only an up event can set the *off* flag to false. Thus the number of off events within $[t_o, t]$ is bounded by $\frac{1}{\lambda}$ plus the number of up events.

Up and Down events. The assumption of Δ gives a trivial bound on those events involving items with very small counts and in particular, those up events immediately following the off events. Such up events are called *poor-up* events or simply *poor-ups*. Using the *off* flag, we can easily adapt the analysis of the simple algorithm to bound all the down and up events of the full algorithm, but except the poor-ups. The following simple observations, derived from Fact 1, allow us to replace Δ with $1/\lambda$ in the previous analysis to obtain a tighter upper bound of $O(\frac{1}{\lambda}\log n)$. Let v be any time in $[t_o, t]$.

- There are at most $1/\lambda$ items whose first event after v is a down event.
- There are at most $1/\lambda$ non-poor-up events after v whose previous event is before v.

It remains to analyze the poor-ups. Consider a poor-up U_j at time v in $[t_o, t]$. By definition, $\mathit{off}_j = \mathit{false}$ at time v. The trick of analyzing U_j's is to consider when the corresponding items will be "off" again instead of what items constitute the up events. Then a characteristic set argument can be formulated easily. Specifically, we first observe that, by Fact 1, there are at most $\frac{1}{\lambda}$ poor-ups whose *off* flags remain false up to time t. Then it remains to consider those U_j whose *off* flags will be set to true at some time $d \le t$. Below we refer to d as the *first off time* of U_j.

Poor-up with early off. Consider a poor-up U_j that occurs at time v in $[t_o, t]$ and has its first off time at d in $[v+1, t]$. Let *F-Expire*(U_j) be all the item j whose first expiry time is within $[v+1, d]$. I.e., *F-Expire*$(U_j) = \sigma_{j,[v+1-W,d-W]}$. As an early off can be due to the expiry of many copies of item j or the arrival of a lot of items, it is natural to divide the poor-ups into two types: with an *absolute* off if $c(d) \le \frac{6}{5}c(v)$, and *relative* off otherwise. For the case with absolute off, we consider the distinct times $t_1, t_2, \ldots, t_k$ in $[t_o, t]$ when such poor-ups occur. Let x_i be the number of such poor-ups at time t_i. Note that $x_i \le 1/(7\lambda)$. For each time t_i, we define the characteristic set

$$F_i = \text{the union of } \textit{F-Expire}(U_j) \text{ over all } U_j \text{ occurring at } t_i, t_{i+1}, \ldots, t_k.$$

Lemma 2.4. **(i)** *For any* $1 \le i \le k-1$, $|F_i| > (1 + x_i\lambda)|F_{i+1}|$. **(ii)** *Within* $[t_o, t]$, *there are* $\sum_{i=1}^{k} x_i = O(\frac{1}{\lambda}\log n)$ *poor-ups each with an absolute off.*

Proof. For (i), consider an item j and a poor-up U_j with an absolute off that occurs at time t_i and has its first off at time d_i. The decrease in c_j must be due to expiry of item j.

$$\begin{aligned}
|\textit{F-Expire}(U_j)| &\ge c_j(t_i) - c_j(d_i) \ge \hat{c}_j(t_i) - \hat{c}_j(d_i) - \lambda c(t_i) - \lambda c(d_i) \\
&> 9\lambda\hat{c}(t_i) - 3\lambda\hat{c}(d_i) - \lambda c(t_i) - \lambda c(d_i) \qquad \text{(definition of up and off)} \\
&\ge (9\lambda(1 - \tfrac{\lambda}{6}) - \lambda)c(t_i) - (3\lambda(1 + \tfrac{\lambda}{6}) + \lambda)c(d_i) \ge 7\lambda c(t_i) - 5\lambda c(d_i) \\
&\ge (7 - 5(\tfrac{6}{5}))\lambda c(t_i) = \lambda c(t_i) \qquad \text{(definition of absolute off)}
\end{aligned}$$

Thus, $|F_i| > |F_{i+1}| + x_i\,(\lambda c(t_i))$. Since $F_i \subseteq \sigma_{[t_i-W+1,t-W]}$, $|F_i| \le c(t_i)$. Therefore, $|F_i| > |F_{i+1}| + x_i\lambda|F_i| > (1 + x_i\lambda)|F_{i+1}|$. By (i), we can prove (ii) similarly to Lemma 2.1 (ii). ∎

Analyzing poor-ups with a relative off is again based on an isolating argument. We divide $[t_o, t]$ into $O(\log n)$ intervals according to how fast the total item count starting from t_o grow; specifically, we want two consecutive time boundaries u_{i-1} and u_i to satisfy

$|\sigma_{[t_o,u_i]}| > \frac{6}{5}|\sigma_{[t_o,u_{i-1}]}|$. Then we show that for any poor-up within $[u_{i-1}, u_i - 1]$, its relative off, if exists, occurs at or after u_i. Thus there are at most $\frac{1}{\lambda}$ such poor-ups within each interval and a total of $O(\frac{1}{\lambda}\log n)$ within $[t_o, t]$.

Lemma 2.5. **(i)** *Consider a poor-up U_j with a relative off. Suppose it occurs at time v in $[t_o, t]$, and its first off time is at d in $[v+1, t]$. Then $|\sigma_{[t_o,d]}| > \frac{6}{5}|\sigma_{[t_o,v]}|$.* **(ii)** *Within $[t_o, t]$, there are at most $O(\frac{1}{\lambda}\log n)$ poor-ups each with a relative off.*

Proof. For (i), by the definition of a relative off, $c(d) > \frac{6}{5}c(v)$. Thus, $|\sigma_{[t_o,d]}| - |\sigma_{[t_o,v]}| = |\sigma_{[v+1,d]}| \geq c(d) - c(v) > \frac{1}{6}c(d) \geq \frac{1}{6}|\sigma_{[t_o,d]}|$. This implies $|\sigma_{[t_o,d]}| > \frac{6}{5}|\sigma_{[t_o,v]}|$.

For (ii), consider the times $t_o = u_0 < u_1 < u_2 < \cdots < u_\ell \leq t$ such that for $i \geq 1$, u_i is the first time such that $|\sigma_{[t_o,u_i]}| > \frac{6}{5}|\sigma_{[t_o,u_{i-1}]}|$. For convenience, let $u_{\ell+1} = t+1$. Note that $|\sigma_{[t_o,t]}| \leq n$ and $\ell = O(\log n)$. Furthermore, for any time $v \in [u_{i-1}, u_i - 1]$, $|\sigma_{[t_o,v]}| \leq \frac{6}{5}|\sigma_{[t_o,u_{i-1}]}|$. Therefore, by (i), for any poor-up of an item j within $[u_{i-1}, u_i - 1]$, its relative off, if exists, occurs at or after u_i, which implies at time $u_i - 1$, $c_j(u_i - 1) \geq \lambda c(u_i - 1)$. Then within each interval $[u_{i-1}, u_i - 1]$, the number of such j as well as the number of poor-ups with a relative off are at most $\frac{1}{\lambda}$. Within $[t_o, t]$, there are $\ell = O(\log n)$ intervals and hence $O(\frac{1}{\lambda}\log n)$ poor-ups each with a relative off. ■

Theorem 2.6. *For approximate counting, each individual stream can use the algorithm AC with $\lambda = \varepsilon/11$ and it sends at most $O(\frac{1}{\varepsilon}\log n)$ words to the root within a window.*

Memory usage of each remote site. Recall that we use two λ-approximate data structures [13, 23] for the total item count and individual item counts, which respectively require $O(\frac{1}{\lambda}\log^2(\lambda n))$ bits and $O(\frac{1}{\lambda})$ words. Note that $O(\frac{1}{\lambda}\log^2(\lambda n))$ bits is equivalent to $O(\frac{1}{\lambda}\log(\lambda n))$ words. Furthermore, at any time, we only need to keep track of the last estimate sent to the root of all item j with *off*$_j$ = *false*, which by Fact 1, requires $O(\frac{1}{\lambda})$ words. By setting $\lambda = \varepsilon/11$ (see Theorem 2.6), the total memory usage of a remote site is $O(\frac{1}{\lambda}\log(\lambda n)) = O(\frac{1}{\varepsilon}\log(\varepsilon n))$ words.

3. Extensions

We extend the previous techniques to solve the problems of frequent items and quantiles and handle out-of-order streams. Below BC refers to our algorithm for basic counting.

Frequent items. Using the algorithms BC and AC, the root can answer the ε-approximate frequent items as follows. Each stream σ communicates with the root using BC with error parameter $\varepsilon/24$ and AC with error parameter $11\varepsilon/24$. At any time t, let $r_\sigma(t)$ and $r_{j,\sigma}(t)$ be the latest estimates of the numbers of all items and item j, respectively, received by the root from σ. To answer a query of frequent items with threshold $\phi \in (0, 1]$ at time t, the root can return all items j with $\sum_\sigma r_{j,\sigma}(t) \geq (\phi - \frac{\varepsilon}{2})\sum_\sigma r_\sigma(t)$ as the set of frequent items.

To see the correctness, let $c_\sigma(t)$ and $c_{j,\sigma}(t)$ be the number of all items and item j in σ at time t, respectively. Algorithm BC guarantees $|r_\sigma(t) - c_\sigma(t)| \leq \frac{\varepsilon}{24}c_\sigma(t)$, and algorithm AC guarantees $|r_{j,\sigma}(t) - c_{j,\sigma}(t)| \leq \frac{11\varepsilon}{24}c_\sigma(t)$. Therefore, if an item j is returned by the root, then $\sum_\sigma c_{j,\sigma}(t) \geq \sum_\sigma r_{j,\sigma}(t) - \frac{11\varepsilon}{24}\sum_\sigma c_\sigma(t) \geq (\phi - \frac{\varepsilon}{2})\sum_\sigma r_\sigma(t) - \frac{11\varepsilon}{24}\sum_\sigma c_\sigma(t) \geq (\phi - \frac{\varepsilon}{2})(1 - \frac{\varepsilon}{24})\sum_\sigma c_\sigma(t) - \frac{11\varepsilon}{24}\sum_\sigma c_\sigma(t) \geq (\phi - \frac{\varepsilon}{2} - \phi\frac{\varepsilon}{24} - \frac{11\varepsilon}{24})\sum_\sigma c_\sigma(t)$ where the second inequality comes from the definition of the algorithm. The last term above is at least

$(\phi - \varepsilon)\sum_\sigma c_\sigma(t)$, so j is a frequent item. If an item j is not returned by the root, then $\sum_\sigma r_{j,\sigma}(t) < (\phi - \frac{\varepsilon}{2})\sum_\sigma r_\sigma(t)$ and we can show similarly that $\sum_\sigma c_{j,\sigma}(t) < \phi \sum_\sigma c_\sigma(t)$.

Quantiles. We give an algorithm for ε-approximate quantiles queries. Let $\lambda = \varepsilon/20$. For each stream, we keep track of the λ-approximate ϕ-quantiles for $\phi = 5\lambda, 10\lambda, 15\lambda, \ldots, 1$. We update the root for all these ϕ-quantiles when one of the following two events occurs: (i) for any k, the value of the $(5k\lambda)$-quantile is larger than the value of the $(5(k+1)\lambda)$-quantile last reported to the root, or (ii) for any k, the value of the $(5k\lambda)$-quantile is smaller than the value of the $(5(k-1)\lambda)$-quantile last reported to the root. The stream also communicates with the root using BC with error parameter λ. In the root's perspective, at any query time t, let $\phi \in (0, 1]$ be the query given and let $r_\sigma(t)$ be the last estimate sent by σ for the number of all items. The root sorts the quantiles last reported by all streams and for each stream σ, gives a weight of $5\lambda r_\sigma(t)$ to each quantile of σ. Then the root returns the smallest item j in the sorted sequence such that the sum of weights for all items no greater than j is at least $\lceil \phi \sum_\sigma r_\sigma(t)\rceil$. Careful counting can show that j is an ε-approximate ϕ-quantile. To bound the communication cost, let n be the number of items of σ arriving or expiring during the window $[t - W + 1, t]$. We observe that when an event occurs, many items have either arrived or expired after the previous event. Using similar analysis as before, we can show that within a window, there are at most $O(\frac{1}{\varepsilon}\log n)$ such events and thus each stream sends $O(\frac{1}{\varepsilon^2}\log n)$ words. By Jensen's inequality again, our algorithm's total communication cost per window is $O(\frac{k}{\varepsilon^2}\log\frac{N}{k})$ where N is the number of items of the k streams that arrive or expire within the window. Note that the lower bound of $O(\frac{1}{\varepsilon}\log(\varepsilon n))$ words for approximate frequent items carries to approximate quantiles, as we can answer approximate frequent items using approximate quantiles as follows. The root poses ε-approximate ϕ-quantile queries for $\phi = \varepsilon, 2\varepsilon, \ldots, 1$. Given the threshold ϕ' for frequent items, the root returns all items that repeatedly occur as $\frac{\phi'}{\varepsilon} - 2$ (or more) consecutive quantiles, and these items are (4ε)-approximate frequent items.

Out-of-order streams. All our algorithms can be extended to out-of-order stream with a communication cost increased by a factor of $\frac{W}{W-\tau}$, as follows. Each stream uses the data structures for out-of-order streams (e.g., [7, 10]) to maintain the local estimates. Then each stream uses our communication algorithms for in-order streams. It is obvious the root can answer the corresponding queries. For the communication cost, consider any time interval $P = [t - (W - \tau) + 1, t]$ of size $W - \tau$. Items arriving in P must have time-stamps in $[t - W + 1, t]$. Using the same arguments as before, we can show the same communication cost of each algorithm, but only for a window of size $W - \tau$ instead of W. Equivalently, in any window of size W, the communication cost is increased by a factor of $O(\frac{W}{W-\tau})$.

References

[1] C. Aggarwal. *Data streams: models and algorithms.* Springer, 2006.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[3] A. Arasu and G. Manku. Approximate counts and quantiles over sliding windows. In *Proc. PODS*, pages 286–296, 2004.

[4] L. Babai, A. Gal, P. Kimmel, and S. Lokam. Communication compleixty of simultaneous messages. *SIAM Journal on Computing*, 33(1):137–166, 2004.

[5] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proc. SODA*, pages 633–634, 2002.

[6] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proc. SIGMOD*, pages 28–39, 2003.

[7] C. Busch and S. Tirthapua. A deterministic algorithm for summarizing asynchronous streams over a sliding window. In *STACS*, 2007.

[8] G. Cormode and M. Garofalakis. Sketching streams through the net: distributed approximate query tracking. In *Proc. VLDB*, pages 13–24, 2005.

[9] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. In *Proc. SIGMOD*, 25–36, 2005.

[10] G. Cormode, F. Korn, and S. Tirthapura. Time-decaying aggregates in out-of-order streams. In *Proc. PODS*, pages 89–98, 2008.

[11] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. In *Proc. SODA*, pages 1076–1085, 2008.

[12] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. Distributed set-expression cardinality estimation. In *Proc. VLDB*, pages 312–323, 2004.

[13] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[14] M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Proc. ESA*, pages 323–334, 2002.

[15] E. Demaine, A. Lopez-Ortiz, and J. Munro. Frequency estimation of internet packet streams with limited space. In *Proc. ESA*, pages 348–360, 2002.

[16] P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *Proc. SPAA*, pages 63–72, 2002.

[17] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proc. PODS*, pages 275–285, 2004.

[18] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proc. STOC*, pages 471–475, 2001.

[19] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc. FOCS*, pages 148–155, 2000.

[20] N. Jain, P. Yalagandula, M. Dahlin, and Y. Zhang. Insight: A distributed monitoring system for tracking continuous queries. In *Proc. SOSP*, pages 1–7, 2005.

[21] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *Proc. SIGMOD*, pages 289–300, 2006.

[22] L. K. Lee and H. F. Ting. Maintaining significant stream statistics over sliding windows. In *Proc. SODA*, pages 724–732, 2006.

[23] L. K. Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proc. PODS*, pages 290–297, 2006.

[24] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *Proc. ICDE*, pages 767–778, 2005.

[25] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *Proc. SIGMOD*, pages 635–646, 2006.

[26] S. Muthukrishnan. *Data streams: algorithms and applications.* Now Publisher Inc., 2005.

[27] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. SIGMOD*, pages 563–574, 2003.

[28] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. *ACM TODS*, 32(4), 2007.

[29] K. Yi and Q. Zhang. Optimal tracking of distributed heavy hitters and quantiles. In *Proc. PODS*, pages 167–174, 2009.

[30] K. Yi and Q. Zhang. Private communication.

ROBUST FAULT TOLERANT UNCAPACITATED FACILITY LOCATION

SHIRI CHECHIK[1] AND DAVID PELEG[1]

[1] Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science
Rehovot 76100, Israel
E-mail address: {shiri.chechik,david.peleg}@weizmann.ac.il

Abstract. In the *uncapacitated facility location* problem, given a graph, a set of demands and opening costs, it is required to find a set of facilities R, so as to minimize the sum of the cost of opening the facilities in R and the cost of assigning all node demands to open facilities. This paper concerns the *robust fault-tolerant* version of the uncapacitated facility location problem (RFTFL). In this problem, one or more facilities might fail, and each demand should be supplied by the closest open facility that did not fail. It is required to find a set of facilities R, so as to minimize the sum of the cost of opening the facilities in R and the cost of assigning all node demands to open facilities that did not fail, after the failure of up to α facilities. We present a polynomial time algorithm that yields a 6.5-approximation for this problem with at most one failure and a $1.5 + 7.5\alpha$-approximation for the problem with at most $\alpha > 1$ failures. We also show that the $RFTFL$ problem is NP-hard even on trees, and even in the case of a single failure.

Introduction

The robust fault-tolerant facility location problem

For a given optimization problem, the robust fault-tolerant version of the problem calls for finding a solution that is still valid even when some components of the system fail. We consider the robust fault-tolerant version of the *uncapacitated facility location (UFL)* problem. In this problem, given a graph G, a demand $\omega(v)$ for every node v and a cost $f(v)$ for opening a facility at v, it is required to find a set of facilities R, so as to minimize the sum of the costs of opening the facilities in R and of shipping the demands of each node from the nearest open facility (at a cost proportional to the distance). In the robust fault-tolerant version of this problem (RFTFL), one or more facilities might fail. Subsequently, each demand should be supplied by the closest open facility that did not fail. It is required to select a set of facilities R, so as to minimize the sum of the costs of opening the facilities in R and the costs of assigning all node demands to open facilities that did not fail, after the failure of up to α facilities. We present a polynomial time algorithm that yields a 6.5-approximation for this problem with at most one failure and a $1.5 + 7.5\alpha$-approximation

Key words and phrases: facility location, approximation algorithms, fault-tolerance.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2454

for the problem with at most $\alpha > 1$ failures. We also show that the $RFTFL$ problem is NP-hard even on trees, and even in the case of a single failure.

Related Work

Many papers deal with approximating the UFL problem, cf. [3, 4, 7, 9, 12, 13]. The best approximation ratio known for this problem is 3/2, shown by Byrka in [2].

A fault-tolerant version of the facility location problem was first introduced by Jain and Vazirani [10], who gave it an approximation algorithm with ratio dependent on the problem parameters. The approximation ratio was later improved by Guha et al. to 2.41 [8] and then by Swamy and Shmoys to 2.076 [14]. However, the variant of the problem studied in these papers is different from the one studied here. In that version, every node j is assigned in advance to a number of open facilities, and pays in advance for all of them. More explicitly, every node j is assigned to r_j open facilities, and its shipping cost is some weighted linear combination of the costs of shipping its demand from all the facilities to which it is assigned. It is required to find a set of facilities R that minimizes the sum of the costs of opening the facilities in R and the sum of costs of shipping the demand of each node j from its r_j facilities in R. This approach is used to capture the expected cost of supplying all clients demand when some of the facilities fail. In contrast, in our definition a node j does not pay in advance for shipping its demand from a number of open facilities. Rather, it pays only for the cost of shipping its demand from the surviving facility that actually supplied its demand. Hence our definition for the fault-tolerant facility location problem requires searching for a set of facilities R that minimizes the sum of the costs of opening the facilities in R and the costs of assigning the demands of each node to one open facility that did not fail, for any failure of up to α facilities. Our approach is used to capture the worst case cost of supplying all clients demand when some of the facilities fail. We argue that our definition may be more natural in some cases, where after the failure of some facilities, each demand should still be supplied by a single supplier, preferably the closest surviving open facility, and each client should pay only for the cost of shipping its demand from that surviving facility, and not for all the other (possibly failed) facilities to which it was assigned originally. On the technical level, the approach taken in [8, 10, 14] is based on applying randomized rounding techniques and primal-dual methods to the corresponding integer linear program. This approach does not readily apply to our version of the problem, and we use a direct combinatorial algorithmic approach instead.

Two other closely related types of problems are the 2-stage stochastic and robust optimization problems (cf. [5, 6]). Both of these models involve two decision stages. In the first stage, some facilities may be purchased. This stage is followed by some scenario depending on the specifics of the problem at hand (in a facility location problem for example, the scenario may specify the clients and their corresponding demands). Subsequently, a second stage is entered, in which it is allowed to purchase additional facilities (whose cost might be much higher than in the first stage). In stochastic optimization there is a distribution over all possible scenarios and the goal is to minimize the expected total cost. In robust optimization the goal is to minimize the cost of the first stage plus the cost of the worst case scenario in the second stage. In contrast with these two models, in our variant the facilities must be selected and opened in advance, and these advance decisions must be adequate under all possible future scenarios.

Billionnet and Costa [1] showed a polynomial time algorithm for solving the ordinary (non-fault-tolerant) UFL problem on trees. In contrast, we show that the fault-tolerant variant RFTFL is NP-hard on trees.

1. Preliminaries

Let us start with common notation to be used later on. Consider an optimization problem Π over a universe V, which given an instance I, requires finding a solution consisting of a set of elements $R \subseteq V$. Denote by $C_\Pi(I,R)$ the cost of the solution R on the instance I of Π. Let $R^*_\Pi(I)$ denote the optimal solution to the problem Π on instance I, and let $C^*_\Pi(I) = C_\Pi(I, R^*_\Pi(I))$ be the cost of the optimal solution. We denote our algorithm for each problem Π studied later by $A_\Pi(I)$. The solution returned by the algorithm is referred to as $R^{alg}_\Pi(I)$ and its cost is $C^{alg}_\Pi(I) = C_\Pi(I, R^{alg}_\Pi(I))$.

Let us now define the *uncapacitated facility location (UFL)* problem. Let $I = \langle G, l, f, \omega \rangle$ be an instance of the problem, where $G = (V, E)$ is a graph with vertex set $V = \{1, ..., n\}$ and edge set E. Each node $v \in V$ hosts a client in need of service, and may host a facility, providing service to clients in nearby nodes. Each edge $e \in E$ has a positive length $l(e)$. The distance $d(u, v)$ between two points u and v on G is defined to be the length of the shortest path between them, where the length of a path is the sum of the lengths of its edges. For each node v, let $f(v)$ denote the *opening cost* associated with placing a facility at v, and let $\omega(v)$ denote the demand of the node v. The *shipping cost* of assigning the demand $\omega(u)$ of a client u to an open facility v is the product $SC_{u,v} = \omega(u)d(u, v)$. The shipping cost $SC_{u,R}$ from a set of open facilities R to a node u is the minimum cost of assigning u to a server in R, namely, $SC_{u,R} = \min\{SC_{u,v} \mid v \in R\}$. Defining the distance $d(v, R)$ between a set of points R and a point v on G to be the minimum distance between v and any node in R, i.e., $d(v, R) = \min_{r \in R} d(v, r)$, we also have $SC_{v,R} = \omega(v)d(v, R)$.

It is required to find a subset $R \subseteq V$ that minimizes the sum of costs of opening the facilities in R and the shipping costs from R to all other nodes. This problem can be formulated as searching for a subset $R \subseteq \{1, ..., n\}$ that minimizes the cost function

$$C_{UFL}(I, R) \ = \ C_{facil}(I, R) + C_{ship}(I, R), \tag{1.1}$$

where

$$C_{facil}(I, R) \ = \ \sum_{r \in R} f(r) \ \text{ and } \ C_{ship}(I, R) \ = \ \sum_{u=1}^{n} SC_{u,R} \ = \ \sum_{u=1}^{n} \omega(u) \cdot d(u, R).$$

Given a set R of open facilities and a facility $r \in R$, let $\varphi(I, r, R)$ denote the set of clients that are served by r under R, i.e., $\varphi(I, r, R) = \{u \mid d(v, r) \leq d(v, r') \ for \ every \ r' \in R\}$, or in other words, the nodes u that satisfy $d(u, R) = d(u, r)$, where ties are broken arbitrarily, i.e., if there is more than one open facility r such that $d(u, R) = d(u, r)$, then just choose one open facility r that satisfies $d(u, R) = d(u, r)$ and add u to $\varphi(I, r, R)$. (When the set R is clear from the context we omit it and write simply $\varphi(I, r)$, or even $\varphi(r)$ when the instance I is clear as well.)

The *robust fault-tolerant facility location (RFTFL)* problem is defined as follows. Each client is supplied by the nearest open facility, and in case this facility fails - it is supplied by the next nearest open facility. We would like to find a solution that is tolerant against a failure of one node. This problem can be formulated as searching for a subset $R \subseteq \{1, ..., n\}$ that minimizes the cost function

$$C_{RFTFL}(I,R) \;=\; C_{facil}(I,R) + C_{ship}(I,R) + C_{backup}(I,R), \tag{1.2}$$

where $C_{facil}(I,R)$ and $C_{ship}(I,R)$ are defined as above and

$$C_{backup}(I,R) \;=\; \max_{r\in R}\left\{\sum_{v\in\varphi(I,r,R)} \omega(v)\cdot(d(v,R\backslash\{r\})-d(v,r))\right\}\;. \tag{1.3}$$

Note that

$$\begin{aligned} C_{RFTFL}(I,R) &= C_{facil}(I,R) + \max_{r\in R}\{C_{ship}(I,R\setminus\{r\})\} \\ &= C_{facil}(I,R) + \max_{r\in R}\left\{\sum_{v=1}^{n} SC_{v,R\backslash\{r\}}\right\} \\ &= C_{facil}(I,R) + \max_{r\in R}\left\{\sum_{v=1}^{n} \omega(v)\cdot d(v,R\backslash\{r\})\right\}\;. \end{aligned} \tag{1.4}$$

Again, when the instance I is clear from the context we omit it and write simply $C_{RFTFL}(R)$, $C_{facil}(R)$, $C_{ship}(R)$, $C_{backup}(R)$, etc.

We also consider the *robust α-fault-tolerant facility location (α_RFTFL)* problem, for integer $\alpha \geq 1$, where the solution should be resilient against a failure of up to α nodes. We define the α_RFTFL as follows. Each client is supplied by the nearest open facility which did not fail. We are looking for a subset $R\subseteq\{1,...,n\}$ that minimizes the cost function

$$C_{\alpha\_RFTFL}(I,R) \;=\; C_{facil}(I,R) + \max_{|R'|\leq\alpha}\left\{\sum_{v=1}^{n} \omega(v)\cdot d(v,R\setminus R')\right\}\;. \tag{1.5}$$

2. A constant approximation algorithm for RFTFL

2.1. The concentrated backup problem and its approximation

Towards developing a constant ratio approximation algorithm for RFTFL, we first consider a different problem, named *concentrated backup (conc_bu)*, defined as follows. An instance of the problem consists of a pair $\langle I,R_1\rangle$ where $I=\langle G,l,f,\omega\rangle$ is defined as before and $R_1=\{r_1,...,r_k\}$ is a set of nodes. In this version, the nodes of R_1 act as both clients and servers (with open facilities), and all other nodes $v\notin R_1$ have zero demands. Informally, it is assumed that we have already paid for opening the facilities in R_1, and each $r\in R_1$ serves itself, at zero shipping cost. The problem requires to assign each client $r\in R_1$ to a backup server $v\neq r$, which may be either some server in R_1 or a new node from $V\backslash R_1$. For a set of nodes R_2, define the *backup cost*

$$C_{bu}(I,R_1,R_2)=\max_{r\in R_1}\left\{SC_{r,R_1\cup R_2\backslash\{r\}}\right\}=\max_{r\in R_1}\{\omega(r)d(r,R_1\cup R_2\backslash\{r\})\}.$$

We are looking for a set R_2 minimizing

$$C_{conc\_bu}(I,R_1,R_2)=C_{facil}(R_2)+C_{bu}(R_1,R_2). \tag{2.1}$$

We denote this minimum cost by $C^*_{conc\_bu}(I,R_1)$. We show a 2-approximation algorithm for the concentrated backup problem.

The problems studied in this section and in section 3.1 are closely related to those considered in [11], and to solve them we use methods similar to the ones presented in [11].

Let us consider a simpler variant of the backup problem, named the *bounded backup (bb)* problem, which is defined on $\langle I, R_1, M\rangle$ and requires looking for a solution R_2 minimizing

$$C_{bb}(I, R_1, M, R_2) = C_{facil}(R_2)$$

subject to the constraint $C_{bu}(R_1, R_2) \leq M$, for integer M. We now present a relaxation algorithm that finds a set R_2 satisfying $C_{facil}(R_2) \leq C^*_{bb}(R_1, M)$ but obeying only the relaxed constraint $C_{bu}(R_1, R_2) \leq 2M$ instead $C_{bu}(R_1, R_2) \leq M$.

```
Algorithm $\boldsymbol{A_{bb}(I, R_1, M)}$
  (1) $R^{alg}_{bb} \leftarrow \emptyset$
  (2) For $i = 1$ to $k$ do:
        • $S_i \leftarrow \{v \mid \omega(r_i)d(v, r_i) \leq 2M\}\backslash\{r_i\}$ /* "relaxed" backup servers for $r_i$ */
        • If $S_i \cap (R_1 \cup R^{alg}_{bb}) = \emptyset$ then add to $R^{alg}_{bb}$ the node $v$ in $S_i$ with the
          minimum facility cost $f(v)$.
  (3) Return $R^{alg}_{bb}$.
```

Let us now prove the properties of algorithm A_{bb}. For every $r_i \in R_1$ let the set of feasible backup servers be $T_i = \{v \mid \omega(r_i)d(v, r_i) \leq M\}\backslash\{r_i\}$. Let the set of relaxed backup servers selected by the algorithm (namely, the final set R^{alg}_{bb} it returns) be $R^{alg}_{bb}(R_1, M) = \{q^{alg}_1, ..., q^{alg}_J\}$. Let ℓ_j be the phase in which the algorithm adds the new facility q^{alg}_j to R^{alg}_{bb}, for $1 \leq j \leq J$.

Lemma 2.1. *$T_{\ell_i} \cap T_{\ell_j} = \emptyset$ for $1 \leq i, j \leq J$.*

Proof: Assume otherwise, and let $v \in T_{\ell_i} \cap T_{\ell_j}$ for some $1 \leq i, j \leq J, i \neq j$. Assume without loss of generality that $\omega(r_{\ell_i}) \leq \omega(r_{\ell_j})$. Since $\omega(r_{\ell_j})d(v, r_{\ell_j}) \leq M$, necessarily $\omega(r_{\ell_i})d(v, r_{\ell_j}) \leq M$ as well, and by the definition of T_{ℓ_i}, also $\omega(r_{\ell_i})d(v, r_{\ell_i}) \leq M$, hence

$$\omega(r_{\ell_i})d(r_{\ell_i}, r_{\ell_j}) \;\leq\; \omega(r_{\ell_i})(d(v, r_{\ell_i}) + d(v, r_{\ell_j})) \;\leq\; 2M,$$

implying that $r_{\ell_j} \in S_{\ell_i} \cap R_1$, so the algorithm should not have opened a new facility in phase ℓ_i, contradiction. ▮

Lemma 2.2. *$C_{facil}(R^{alg}_{bb}(R_1, M)) \leq C^*_{bb}(R_1, M)$.*

Proof: Notice that there must be at least one node from every T_i in the optimal solution $R^*_{bb}(R_1, M)$. By Lemma 2.1 the sets $T_{\ell_1}, ..., T_{\ell_J}$ are disjoint, so there are at least J distinct nodes $q^*_j \in R^*_{bb}(R_1, M)$, one from each T_{ℓ_j}, for $1 \leq j \leq J$. In each phase i, the algorithm selects the cheapest node in $S_i \supseteq T_i$. Therefore, $f(q^{alg}_j) \leq f(q^*_j)$ for every $1 \leq j \leq J$. Hence $C_{facil}(R^{alg}_{bb}(R_1, M)) = \sum\limits_{j=1}^{J} f(q^{alg}_j) \leq \sum\limits_{j=1}^{J} f(q^*_j) \leq C^*_{bb}(R_1, M)$. ▮

Lemma 2.3. *$C_{bu}(R_1, R^{alg}_{bb}(R_1, M)) \leq 2M$.*

Proof: For each server r_i in R_1, the algorithm ensures that there is at least one open facility from the set S_i, so $\omega(r_i)d(r_i, R_1 \cup R^{alg}_{bb}(R_1, M) \setminus \{r_i\}) \leq 2M$. ▮

Now we present an approximation algorithm A_{conc_bu} for the concentrated backup problem using the relaxation algorithm A_{bb} for the bounded backup problem. First note that there can be at most nk possible values for the shipping costs $SC_{u,v} = \omega(u)d(u,v)$.

```
Algorithm $\boldsymbol{A_{conc\_bu}(I, R_1)}$
  (1) For every $M \in \{SC_{u,v} \mid u, v \in V\}$ do:
        • let $R^{alg}_{bb}(R_1, M) \leftarrow A_{bb}(I, R_1, M)$.
  (2) Return the set $R^{alg}_{bb}(R_1, M)$ with the minimum cost
      $C_{conc\_bu}(R_1, R^{alg}_{bb}(R_1, M))$.
```

Lemma 2.4. $C^{alg}_{conc\_bu}(I, R_1) \leq 2C^*_{conc\_bu}(I, R_1)$.

Proof: Recall that, letting $R^*_2 = R^*_{conc\_bu}(R_1)$,

$$C^*_{conc\_bu}(I, R_1) \;=\; C_{conc\_bu}(I, R_1, R^*_2) \;=\; C_{facil}(R^*_2) + C_{bu}(I, R_1, R^*_2).$$

Let $u \in R_1$ be the node that attains the maximum shipping cost $SC_{u,R_1 \cup R_2 \setminus \{u\}}$, i.e., satisfies $\omega(u)d(u, R_1 \cup R^*_2 \setminus \{u\}) = C_{bu}(I, R_1, R^*_2)$, and let $v \in R_1 \cup R^*_2 \setminus \{u\}$ be its backup, i.e., the closest node to u. Then $C^*_{conc\_bu}(I, R_1) = C_{conc\_bu}(I, R_1, R^*_2) = C_{facil}(R^*_2) + SC_{u,v}$. Since the algorithm examines all possible values of M, it tests also $M_0 = SC_{u,v}$. For this value, the returned set $R^{alg}_{bb}(R_1, M_0)$ has opening cost at most $C^*_{bb}(R_1, M_0) = C_{facil}(R^*_2)$ and backup cost at most
$C_{bu}(I, R_1, R^{alg}_{bb}(R_1, M_0)) \leq 2M_0$ by Lemmas 2.2 and 2.3. Since the algorithm takes the minimum cost $C_{conc\_bu}(R_1, R^{alg}_{bb}(R_1, M))$ over all possible values of M, the resulting cost satisfies $C^{alg}_{conc\_bu}(I, R_1) \leq C_{facil}(R^*_2) + 2SC_{u,v} \leq 2C^*_{conc\_bu}(I, R_1)$, namely, an approximation ratio of 2. ▮

2.2. 6.5-approximation algorithm for RFTFL

We now present a polynomial time algorithm A_{RFTFL} that yields 6.5-approximation for the robust fault-tolerant uncapacitated facility location problem RFTFL. Consider an instance $I = \langle G, l, f, \omega \rangle$ of the problem. The algorithm consists of three stages.

Stage 1: Apply the 1.5-approximation algorithm of [2] to the original UFL problem in order to find an initial subset R_1 of servers. Notice that the cost of this solution satisfies

$$C_{UFL}(R_1) \leq 1.5C^*_{UFL} \leq 1.5C^*_{RFTFL}\ . \tag{2.2}$$

Each node is now assigned to a server in R_1. Next, we need to assign to each node a *backup server* which will serve it in case its original server fails.

Stage 2: Transform the given instance $I = \langle V, l, \omega, f \rangle$ of the problem into an instance $I' = \langle V, l, \omega', f' \rangle$ as follows. First, change the facility cost f by setting $f'(r) = 0$ for $r \in R_1$. Next, for each server $r \in R_1$, relocate all the demands of the nodes that are served by r, and place them at the server r itself, that is, set

$$\omega'(r) = \begin{cases} \sum\limits_{v \in \varphi(I,r,R_1)} \omega(v), & for\ r \in R_1, \\ 0, & for\ r \notin R_1. \end{cases} \tag{2.3}$$

Stage 3: Invoke the 2-approximation algorithm A_{conc_bu} for the concentrated backup problem on the new instance I' and the set R_1. The approximation algorithm returns a new set R_2. We then return the set $R_1 \cup R_2$ as the final set of open facilities.

Lemma 2.5. *For every instance I and set $R_1 \subseteq V$, $C^*_{conc\_bu}(I', R_1) \leq C^*_{RFTFL}(I) + C_{UFL}(I, R_1)$.*

Proof: Consider some vertex $r \in R_1$ and let $\varphi(I, r, R_1) = \{v_1^r, ..., v_{k_r}^r\}$ be the nodes it serves. Consider the optimal solution $R^*_{RFTFL}(I)$ to the RFTFL problem. Let d_i^r be the distance from r to v_i^r for $1 \leq i \leq k_r$, and also let x_i^r be the distance from v_i^r to its optimal backup server, which is also its distance to $R_r^* \equiv R_1 \cup R^*_{RFTFL}(I) \backslash \{r\}$, i.e., $x_i^r = d(v_i^r, R_r^*)$. By the triangle inequality, $d(r, R_r^*) \leq d_i^r + x_i^r$, for every $1 \leq i \leq k_r$, so

$$\begin{aligned} \omega'(r) \cdot d(r, R_r^*) &= \sum_{l=1}^{k_r} \omega(v_l^r) \cdot d(r, R_r^*) \leq \sum_{l=1}^{k_r} \omega(v_l^r)(d_l^r + x_l^r) \\ &= \sum_{l=1}^{k_r} \omega(v_l^r) d(v_l^r, R_1) + \sum_{l=1}^{k_r} \omega(v_l^r) x_l^r \\ &\leq \sum_{v=1}^{n} \omega(v) \cdot d(v, R_1) + \sum_{v=1}^{n} \omega(v) \cdot d(v, R_r^*). \end{aligned}$$

Therefore,

$$\begin{aligned} C_{bu}(I', R_1, R^*_{RFTFL}(I)) &= \max_{r \in R_1} \{\omega'(r) \cdot d(r, R_r^*)\} \\ &\leq C_{ship}(I, R_1) + \max_{r \in R_1} \left\{ \sum_{v=1}^{n} \omega(v) \cdot d(v, R_r^*) \right\}. \end{aligned}$$

Using (1.4) and (2.1) we now bound the cost of the optimal solution for problem *conc_bu* by

$$\begin{aligned} C^*_{conc\_bu}(I', R_1) &\leq C_{conc\_bu}(I', R_1, R^*_{RFTFL}(I)) \\ &= C_{facil}(I', R^*_{RFTFL}(I)) + C_{bu}(I', R_1, R^*_{RFTFL}(I)) \\ &\leq C_{facil}(I', R^*_{RFTFL}(I)) + \max_{r \in R_1} \left\{ \sum_{v=1}^{n} \omega(v) d(v, R_r^*) \right\} + C_{ship}(I, R_1) \\ &\leq C^*_{RFTFL}(I) + C_{ship}(I, R_1) \leq C^*_{RFTFL}(I, R_1) + C_{UFL}(I, R_1). \end{aligned}$$ ∎

Lemma 2.6. *For every instance I and sets $R_1, R_2 \subseteq V$,*
$C_{RFTFL}(I, R_1 \cup R_2) \leq C_{UFL}(I, R_1) + C_{conc\_bu}(I', R_1, R_2)$.

Proof: The cost of opening the facilities in $R_1 \cup R_2$ is clearly at most the cost of opening the facilities in R_1 plus the cost of opening the facilities in R_2. For every facility $r \in R_1 \cup R_2$, in order to bound $C_{ship}(I, R_1 \cup R_2 \setminus \{r\})$, note that one can first move each client v to its closest open facility in R_1, and then move all the clients assigned to r (if $r \in R_1$) to the backup facility of r in R_2. The inequality follows. More formally we have the following. Recall that by (1.4),

$$C_{RFTFL}(I, R_1 \cup R_2) = C_{facil}(I, R_1 \cup R_2) + \max_{r \in R_1 \cup R_2} \{ C_{ship}(I, R_1 \cup R_2 \setminus \{r\}) \}.$$

Consider first the case that $\max_{r\in R_1\cup R_2}\{C_{ship}(I,R_1\cup R_2\setminus\{r\})\}$ is attained for some $r'\in R_2$. In this case, we get by (1.1) that

$$\begin{aligned}
C_{RFTFL}(I,R_1\cup R_2) &= C_{facil}(I,R_1\cup R_2)+C_{ship}(I,R_1\cup R_2\setminus\{r'\})\\
&\le C_{facil}(I,R_1\cup R_2)+C_{ship}(I,R_1)\\
&= C_{UFL}(I,R_1)+C_{facil}(I,R_2)\le C_{UFL}(I,R_1)+C_{conc\_bu}(I',R_1,R_2).
\end{aligned}$$

So now assume that $\max_{r\in R_1\cup R_2}\{C_{ship}(I,R_1\cup R_2\setminus\{r\})\}$ is attained for some $r'\in R_1$. Therefore,

$$\begin{aligned}
C_{RFTFL}(I,R_1\cup R_2) &= C_{facil}(I,R_1\cup R_2)+C_{ship}(I,R_1\cup R_2\setminus\{r'\})\\
&= C_{facil}(I,R_1)+C_{facil}(I,R_2)+\sum_{v=1}^{n}SC_{v,R_1\cup R_2}\\
&\quad+\sum_{v\in\varphi(I,r',R_1\cup R_2)}\omega(v)\cdot(d(v,R_1\cup R_2\setminus\{r'\})-d(v,r'))\\
&\le C_{UFL}(I,R_1)+C_{facil}(I,R_2)\\
&\quad+\max_{r\in R_1}\left\{\sum_{v\in\varphi(I,r,R_1)}\omega(v)\cdot(d(r,R_1\cup R_2\setminus\{r\}))\right\}\\
&= C_{UFL}(I,R_1)+C_{facil}(I,R_2)+\max_{r\in R_1}\left\{w'(r)\cdot(d(r,R_1\cup R_2\setminus\{r\}))\right\}\\
&= C_{UFL}(I,R_1)+C_{conc\_bu}(I',R_1,R_2). \qquad ▮
\end{aligned}$$

Lemma 2.7. *Algorithm A_{RFTFL} yields a 6.5-approximation for the RFTFL problem.*

Proof: Consider the set of opened facilities $R_1\cup R_2$. By Lemma 2.4, R_2 is a 2-approximation of the concentrated backup problem on the instance I', so

$$C_{conc\_bu}(I',R_1,R_2)\le 2C^*_{conc\_bu}(I',R_1).$$

By Lemma 2.5, $C^*_{conc\_bu}(I',R_1)\le C^*_{RFTFL}(I)+C_{UFL}(I,R_1)$, hence

$$C_{conc\_bu}(I',R_1,R_2)\le 2C^*_{RFTFL}(I)+2C_{UFL}(I,R_1).$$

Using Lemma 2.6 we get

$$C_{RFTFL}(I,R_1\cup R_2)\ \le\ 3C_{UFL}(I,R_1)+2C^*_{RFTFL}(I),$$

and by (2.2), $C_{RFTFL}(I,R_1\cup R_2)\ \le\ 6.5C^*_{RFTFL}(I)$. ▮

3. An approximation algorithm for α_RFTFL

3.1. The concentrated α_backup problem

As in the case of a single failure, we first consider a different problem, named *concentrated α_backup (conc_α_bu)*, defined as follows. An instance of the problem consists of a pair $\langle I,R_1\rangle$ where $I=\langle G,l,f,\omega\rangle$ is defined as before and R_1 is a set of nodes. The nodes of R_1 act as both clients and servers (with open facilities), and all other nodes $v\notin R_1$ have zero demands. We are looking for a set R_2 minimizing

$$C_{conc\_\alpha\_bu}(I,R_1,R_2)=C_{facil}(R_2)+C_{\alpha\_bu}(I,R_1,R_2), \tag{3.1}$$

where C_{α_bu} is the *maximum α_backup cost* for a set of nodes R_2, defined as

$$C_{\alpha\_bu}(I, R_1, R_2) = \max_{|F| \le \alpha} \left\{ \sum_{r \in (F \cap R_1)} \omega(r) \cdot d(r, R_1 \cup R_2 \backslash F) \right\}.$$

We will shortly present a 3α-approximation algorithm for the concentrated α-backup problem.

Towards this, let us first consider a simpler variant of the backup problem, named the *α-bounded backup (α_bb)* problem, which is defined on $\langle I, R_1, M \rangle$ and requires looking for a solution R_2 minimizing

$$C_{\alpha\_bb}(R_1, M, R_2) = C_{facil}(R_2)$$

subject to the constraint $C_{light\_\alpha\_bu}(R_1, R_2) \le M$ for some integer M, where

$$C_{light\_\alpha\_bu}(R_1, R_2) = \max_{r \in R_1, |F| \le \alpha} \{\omega(r) d(r, R_1 \cup R_2 \backslash F)\}.$$

We now present a relaxation algorithm that finds a set R_2 satisfying $C_{facil}(R_2) \le C^*_{\alpha\_bb}(R_1, M)$ but allowing the relaxed constraint $C_{light\_\alpha\_bu}(R_1, R_2) \le 3M$ instead of $C_{light\_\alpha\_bu}(R_1, R_2) \le M$.

```
Algorithm $\boldsymbol{A_{\alpha\_bb}(I, R_1, M)}$
  (1) $R^{alg}_{\alpha\_bb} \leftarrow \emptyset$
  (2) Let $r_1, ..., r_k$ be the servers in $R_1$ sorted by nonincreasing order of demands.
  (3) $Z \leftarrow \emptyset$ /* The set of servers $r_i$ where the algorithm opens facilities in phase
      $i$ */
  (4) For $i = 1$ to $k$ do:
  (5)     • $S_i \leftarrow \{v \mid \omega(r_i) d(v, r_i) \le 2M\} \backslash \{r_i\}$.
          • $T_i \leftarrow \{v \mid \omega(r_i) d(v, r_i) \le M\} \backslash \{r_i\}$
          • If $S_i \cap Z = \emptyset$ then:
              – Add to $R^{alg}_{\alpha\_bb}$, the $\alpha - |T_i \cap (R_1 \cup R^{alg}_{\alpha\_bb})|$ nodes in $T_i \backslash (R_1 \cup R^{alg}_{\alpha\_bb})$
                with the lowest facility costs.
              – $Z \leftarrow Z \cup \{r_i\}$
  (6) Return $R^{alg}_{\alpha\_bb}$.
```

Let us now prove the properties of Alg. A_{α_bb}. Let $\{\ell_j \mid 1 \le j \le J\}$ be the phases in which the algorithm adds new facilities to $R^{alg}_{\alpha\_bb}$. By a proof similar to that of Lemma 2.1, we have the following.

Lemma 3.1. *$T_{\ell_i} \cap T_{\ell_j} = \emptyset$ for $1 \le j < i \le J$.*

Lemma 3.2. *$C_{facil}(R^{alg}_{\alpha\_bb}(R_1, M)) \le C^*_{\alpha\_bb}(R_1, M)$.*

Proof: There must be at least α nodes in every T_{ℓ_j} in the optimal solution $R^*_{\alpha\_bb}(R_1, M)$. By Lemma 3.1 the sets T_{ℓ_j} for $1 \le j \le J$ are disjoint, so the only nodes that the algorithm adds to $R^{alg}_{\alpha\_bb}$ from the set T_{ℓ_j} are added at phase ℓ_j. The algorithm selects the cheapest nodes in T_{ℓ_j} in order to complete to α nodes. Therefore, $C_{facil}(R^{alg}_{\alpha\_bb}(R_1, M) \cap T_{\ell_j}) \le$

$C_{facil}(R^*_{\alpha\_bb}(R_1,M)\cap T_{\ell_j})$ for every $1\le j\le J$. Hence

$$\begin{aligned} C_{facil}(R^{alg}_{\alpha\_bb}(R_1,M)) &= \sum_{j=1}^{J} C_{facil}(R^{alg}_{\alpha\_bb}(R_1,M)\cap T_{\ell_j}) \le \sum_{j=1}^{J} C_{facil}(R^*_{\alpha\_bb}(R_1,M)\cap T_{\ell_j}) \\ &\le C^*_{\alpha\_bb}(R_1,M). \quad \blacksquare \end{aligned}$$

Lemma 3.3. $C_{\alpha\_bu}(R_1,R^{alg}_{\alpha\_bb}(R_1,M))\le 3M$.

Proof: For each server $v_i\in R_1$, the algorithm ensures that either there are at least α open facilities from the set T_i or v_i is at distance at most $2M$ from another $v_j\in R_1$ that has α open facilities from the set T_j. In the first case the distance is at most M and in the second - at most $3M$. ▮

Now we present an approximation algorithm $A_{conc\_\alpha\_bu}$ for the concentrated α_backup problem, using the relaxation algorithm A_{α_bb} for the α_bounded backup problem.

```
Algorithm $\boldsymbol{A_{conc\_\alpha\_bu}(I,R_1)}$
  (1) For every subset $T\subseteq\{SC_{v,u}\mid v,u\in V\}$ such that $|T|\le\alpha$ do:
        • $M(T)\leftarrow\sum_{m\in T} m$
        • let $R^{alg}_{\alpha\_bb}(R_1,M(T))\leftarrow A_{\alpha\_bb}(I,R_1,M(T))$.
  (2) Return the set $R^{alg}_{\alpha\_bb}(R_1,M(T))$ with the minimum cost
      $C_{conc\_\alpha\_bu}(R_1,R^{alg}_{\alpha\_bb}(R_1,M(T)))$.
```

Lemma 3.4. $C^{alg}_{conc\_\alpha\_bu}(I,R_1)\le 3\alpha C^*_{conc\_\alpha\_bu}(I,R_1)$.

Proof: Denote the optimal solution for $conc\_\alpha\_bu$ on $\langle I,R_1\rangle$ by $R^*_2=R^*_{conc\_\alpha\_bu}(R_1)$. Then

$$C^*_{conc\_\alpha\_bu}(I,R_1)=C_{conc\_\alpha\_bu}(I,R_1,R^*_2)=C_{facil}(R^*_2)+C_{\alpha\_bu}(I,R_1,R^*_2).$$

Let $\{u_1,...,u_j\}\subseteq R_1$ and $\{v_1,...,v_j\}\subseteq R_1\cup R^*_2$ for some $j\le\alpha$ be the sets of nodes that attain the maximum shipping cost, i.e., satisfy $C_{\alpha\_bu}(I,R_1,R^*_2)=M_0$ for $M_0=\sum_{i=1}^{j}SC_{u_i,v_i}=\sum_{i=1}^{j}\omega(u_i)d(u_i,v_i)$. Then $C^*_{conc\_\alpha\_bu}(I,R_1)=C_{facil}(R^*_2)+M_0$. Notice that there must be at least α nodes in the set $R^*_2\cup R_1$ at distance at most M_0 from every server r in R_1. Clearly $C_{facil}(R^*_{\alpha\_bb}(R_1,M_0))\le C_{facil}(R^*_2)$. Since the algorithm examines all possible values of $M(T)$, it tests also M_0. For this value, the returned set $R^{alg}_{\alpha\_bb}(R_1,M_0)$ has opening cost at most $C^*_{\alpha\_bb}(R_1,M_0)\le C_{facil}(R^*_2)$ and backup cost at most $C_{\alpha\_bu}(I,R_1,R^{alg}_{\alpha\_bb}(R_1,M_0))\le 3M_0$ by Lemmas 3.2 and 3.3. Since the algorithm takes the minimum cost $C_{conc\_\alpha\_bu}(R_1,R^{alg}_{\alpha\_bb}(I,R_1,M(T)))$ over all possible subsets T, the resulting cost is at most

$$\begin{aligned} C^{alg}_{conc\_\alpha\_bu}(I,R_1) &\le C_{conc\_\alpha\_bu}(I,R_1,R^{alg}_{\alpha\_bb}(R_1,M_0)) \\ &\le C_{facil}(R^*_2)+\max_{|F|\le\alpha}\left\{\sum_{r\in(F\cap R_1)}\omega(r)d(r,R_1\cup R^{alg}_{\alpha\_bb}(R_1,M_0)\backslash F)\right\} \\ &\le C_{facil}(R^*_2)+3\alpha M_0 \le 3\alpha C^*_{conc\_\alpha\_bu}(I,R_1). \quad \blacksquare \end{aligned}$$

3.2. $(1.5 + 7.5\alpha)$-approximation algorithm to the α_RFTFL

We now present a polynomial time algorithm named A_{α_RFTFL}, yielding a $(1.5 + 7.5\alpha)$-approximation for the robust fault-tolerant uncapacitated facility location problem α_RFTFL against a failure of α nodes, for constant $\alpha > 1$. Consider an instance $I = \langle G, l, f, \omega \rangle$ of the problem. The algorithm is similar to Algorithm RFTFL, except for the third stage. Instead of invoking the 2-approximation algorithm A_{conc_bu} for the concentrated backup problem on the new instance I' and the set R_1, invoke the 3α-approximation algorithm $A_{conc\_\alpha\_bu}$ for the concentrated α_backup problem on the new instance I' and the set R_1. Algorithm $A_{conc\_\alpha\_bu}$ returns a new set R_2^{alg}. Algorithm A_{α_RFTFL} now returns the set $R_1 \cup R_2^{alg}$. Proof of the following lemma is deferred to the full paper.

Lemma 3.5. *Algorithm A_{α_RFTFL} yields a $(1.5 + 7.5\alpha)$-approximation for the α_RFTFL problem.*

4. Robust Fault-tolerant uncapacitated facility location on trees

In this section we show that the RFTFL problem is NP-hard even on trees. The claim holds even in the case where only the edge lengths or only the node demands are variable and the other parameters are uniform. An instance of the RFTFL problem is $\langle T, l, f, \omega, P \rangle$, where T is a tree, l, f and ω are defined as before and P is an integer. It is required to decide if the cost of the optimal solution to the RFTFL problem on the instance $\langle T, l, f, \omega \rangle$ is P or less.

The proofs, via reductions from subset sum and from a variant of the partition problem, are deferred to the full paper. The following results are established.

Theorem 4.1. *$RFTFL$ on trees is NP-complete even with*

(1) *unit edge lengths and opening costs (but variable node demands),*
(2) *unit node demands and opening costs (but variable edge lengths).*

References

[1] A. Billionnet and M. Costa, Solving the uncapacited plant location problem on trees, *Discrete Applied Mathematics* **49**, (1994), 51–59.

[2] J. Byrka, An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In *Proc. 10th Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2007, 29–43.

[3] M. Charikar and S. Guha, Improved combinatorial algorithms for facility location problems, *SIAM J. Comput.* **34**, (2005), 803–824.

[4] F. A. Chudak and D. B. Shmoys, Improved approximation algorithms for the uncapacitated facility location problem, *SIAM J. Comput.* **33**, (2003) 1–25.

[5] K. Dhamdhere, V. Goyal, R. Ravi, and M. Singh, How to Pay, Come What May: Approximation Algorithms for Demand-Robust Covering Problems, In *Proc. 46th IEEE Symp. on Foundations of Computer Science (FOCS)*, 2005, 367–378.

[6] U. Feige, K. Jain, M. Mahdian, and V.S. Mirrokni, Robust combinatorial optimization with exponential scenarios, In *Proc. 12th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, Ithaca, NY, 2007, 439–453.

[7] S. Guha and S. Khuller, Greedy strikes back: Improved facility location algorithms, *J. of Algorithms* **31**, (1999), 228–248.

[8] Sudipto Guha, Adam Meyerson, and Kamesh Munagala, A constant factor approximation algorithm for the fault-tolerant facility location problem, *J. Algorithms* **48** ,(2003) , 429–440.

[9] K. Jain, M. Mahdian, and A. Saberi, A new greedy approach for facility location problems, In *Proc. 34th ACM Symposium on Theory of Computing (STOC)*, 2002, 731–740.

[10] K. Jain, V. Vazirani, An approximation algorithm for the fault tolerant metric facility location problem, In *Proc. APPROX*,LNCS Vol. 1913, 2000, 177–183.

[11] S. Khuller, R. Pless, and Y. Sussmann, Fault tolerant k-center problems. *Theoret. Comput. Sci.* 242 12 (2000), 237-245

[12] M. Mahdian, Y. Ye, and J. Zhang, Approximation algorithms for metric facility location problems, *SIAM J. on Computing* **36**, (2006), 411–432.

[13] D.B. Shmoys, Approximation algorithms for facility location problems, In *Proc. 3rd Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, LNCS, Vol. 1913, (2000), 265–274.

[14] C. Swamy, D. Shmoys, Fault-tolerant facility location, In *Proc. 14th ACM-SIAM SODA*, 2003, 735–736.

[15] L.A. Wolsey, An analysis of the greedy algorithm for the submodular set covering problem, *Combinatorica* **2**, (1982), 385–393.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 203-214
www.stacs-conf.org

EFFICIENT AND ERROR-CORRECTING DATA STRUCTURES FOR MEMBERSHIP AND POLYNOMIAL EVALUATION

VICTOR CHEN [1] AND ELENA GRIGORESCU [2] AND RONALD DE WOLF [3]

[1] Tsinghua University ITCS and MIT CSAIL
E-mail address: victor.vc@gmail.com

[2] MIT CSAIL
E-mail address: elena_g@mit.edu

[3] CWI. Science Park 123. 1098XG Amsterdam. The Netherlands
E-mail address: rdewolf@cwi.nl

ABSTRACT. We construct efficient data structures that are resilient against a constant fraction of adversarial noise. Our model requires that the decoder answers *most* queries correctly with high probability and for the remaining queries, the decoder with high probability either answers correctly or declares “don’t know.” Furthermore, if there is no noise on the data structure, it answers *all* queries correctly with high probability. Our model is the common generalization of an error-correcting data structure model proposed recently by de Wolf, and the notion of “relaxed locally decodable codes” developed in the PCP literature.

We measure the efficiency of a data structure in terms of its *length* (the number of bits in its representation), and query-answering time, measured by the number of *bit-probes* to the (possibly corrupted) representation. We obtain results for the following two data structure problems:

- (Membership) Store a subset S of size at most s from a universe of size n such that membership queries can be answered efficiently, i.e., decide if a given element from the universe is in S.
  We construct an error-correcting data structure for this problem with length nearly linear in $s \log n$ that answers membership queries with $O(1)$ bit-probes. This nearly matches the asymptotically optimal parameters for the noiseless case: length $O(s \log n)$ and one bit-probe, due to Buhrman, Miltersen, Radhakrishnan, and Venkatesh.
- (Univariate polynomial evaluation) Store a univariate polynomial g of degree $\deg(g) \leq s$ over the integers modulo n such that evaluation queries can be answered efficiently, i.e., we can evaluate the output of g on a given integer modulo n.
  We construct an error-correcting data structure for this problem with length nearly linear in $s \log n$ that answers evaluation queries with $\operatorname{polylog} s \cdot \log^{1+o(1)} n$ bit-probes. This nearly matches the parameters of the best-known noiseless construction, due to Kedlaya and Umans.

1998 ACM Subject Classification: E1, E4.

Key words and phrases: Data Structures, Error-Correcting Codes, Membership, Polynomial Evaluation.

This work was done when the author was a student at MIT. Supported by NSF award CCF-0829672 and National Natural Science Foundation of China Grant 60553001, the National Basic Research Program of China Grant 2007CB807900, 2007CB807901.

This work started when this author was visiting CWI in Summer 2008. Supported by NSF award CCF-0829672.

Supported by a Vidi grant from the Netherlands Organization for Scientific Research (NWO).

SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2455

1. Introduction

The area of data structures is one of the oldest and most fundamental parts of computer science, in theory as well as in practice. The underlying question is a time-space tradeoff: we are given a piece of data, and we would like to store it in a short, space-efficient data structure that allows us to quickly answer specific queries about the stored data. On one extreme, we can store the data as just a list of the correct answers to all possible queries. This is extremely time-efficient (one can immediately look up the correct answer without doing any computation) but usually takes significantly more space than the information-theoretic minimum. At the other extreme, we can store a maximally compressed version of the data. This method is extremely space-efficient but not very time-efficient since one usually has to undo the whole compression first. A good data structure sits somewhere in the middle: it does not use much more space than the information-theoretic minimum, but it also stores the data in a structured way that enables efficient query-answering.

It is reasonable to assume that most practical implementations of data storage are susceptible to *noise*: over time some of the information in the data structure may be corrupted or erased by various accidental or malicious causes. This buildup of errors may cause the data structure to deteriorate so that most queries are not answered correctly anymore. Accordingly, it is a natural task to design data structures that are not only efficient in space and time but also resilient against a certain amount of *adversarial* noise, where the noise can be placed in positions that make decoding as difficult as possible.

Ways to protect information and computation against noise have been well studied in the theory of error-correcting codes and of fault-tolerant computation. In the data structure literature, constructions under often incomparable models have been designed to cope with noise. We mention a few of these models here. First, Aumann and Bender [1] studied pointer-based data structures such as linked lists, stacks, and binary search trees. In this model, errors (adversarial but detectable) occur whenever all the pointers from a node are lost. They studied the dependence between the number of errors and the number of nodes that become irretrievable, and designed a number of efficient data structures where this dependence is reasonable.

Another model for studying data structures with noise is the faulty-memory RAM model, introduced by Finocchi and Italiano [10]. In a faulty-memory RAM, there are $O(1)$ memory cells that cannot be corrupted by noise. Elsewhere, errors (adversarial and undetectable) may occur at any time, even during the decoding procedure. Many data structure problems have been examined in this model, such as sorting [8], searching [9], priority queues [13] and dictionaries [4]. However, the number of errors that can be tolerated is typically less than a linear portion of the size of the input. Furthermore, correctness can only be guaranteed for keys that are not affected by noise. For instance, for the problem of comparison-sorting on n keys, the authors of [8] designed a resilient sorting algorithm that tolerates $\sqrt{n \log n}$ keys being corrupted and ensures that the set of uncorrupted keys remains sorted.

Recently, de Wolf [19] considered another model of resilient data structures. The representation of the data structure is viewed as a bit-string, from which a decoding procedure can read any particular set of bits to answer a data query. The representation must be able to tolerate a constant fraction δ of adversarial noise in the bit-string[1] (but not inside the decoding procedure). His model generalizes the usual noise-free data structures (where $\delta = 0$) as well as the so-called "locally decodable codes" (LDCs) [14]. Informally, an LDC is an encoding that is tolerant of noise and allows

[1] We only consider bit-flip-errors here, not erasures. Since erasures are easier to deal with than bit-flips, it suffices to design a data structure dealing with bit-flip-errors.

fast decoding so that each message symbol can be retrieved correctly with high probability. Using LDCs as building blocks, de Wolf constructed data structures for several problems.

Unfortunately, de Wolf's model has the drawback that the optimal time-space tradeoffs are much worse than in the noise-free model. The reason is that all known constructions of LDCs that make $O(1)$ bit-probes [21, 7] have very poor encoding length (super-polynomial in the message length). In fact, this encoding length provably must be super-linear in the message length [14, 16, 20]. As his model is a generalization of LDCs, data structures cannot have a succinct representation that has length proportional to the information-theoretic bound.

We thus ask: what is a clean model of data structures that allows efficient representations *and* has error-correcting capabilities? Compared with the pointer-based model and the faulty-memory RAM, de Wolf's model imposes a rather stringent requirement on decoding: *every* query must be answered correctly with high probability from the possibly corrupted encoding. While this requirement is crucial in the definition of LDCs due to their connection to complexity theory and cryptography, for data structures it seems somewhat restrictive.

In this paper we consider a broader, more relaxed notion of error-correction for data structures. In our model, for most queries, the decoder has to return the correct answer with high probability. However, for the few remaining queries, the decoder may claim ignorance, i.e., declare the data item unrecoverable from the (corrupted) data structure. Still, for *every* query, the answer is incorrect only with small probability. In fact, just as de Wolf's model is a generalization of LDCs, our model in this paper is a generalization of the "relaxed" locally decodable codes (RLDCs) introduced by Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan [3]. They relax the usual definition of an LDC by requiring the decoder to return the correct answer on *most* rather than all queries. For the remaining queries it is allowed to claim ignorance, i.e., to output a special symbol '$\perp$' interpreted as "don't know" or "unrecoverable." As shown in [3], relaxing the LDC-definition like this allows for constructions of RLDCs with $O(1)$ bit-probes of *nearly linear* length.

Using RLDCs as building blocks, we construct error-correcting data structures that are very efficient in terms of time as well as space. Before we describe our results, let us define our model formally. First, a *data structure problem* is specified by a set D of *data items*, a set Q of *queries*, a set A of *answers*, and a function $f : D \times Q \to A$ which specifies the correct answer $f(x,q)$ of query q to data item x. A data structure for f is specified by four parameters: t the number bit-probes, δ the fraction of noise, ε an upper bound on the error probability for each query, and λ an upper bound on the fraction of queries in Q that are not answered correctly with high probability (the 'λ' stands for "lost").

Definition 1.1. Let $f : D \times Q \to A$ be a data structure problem. Let $t > 0$ be an integer, $\delta \in [0,1]$, $\varepsilon \in [0,1/2]$, and $\lambda \in [0,1]$. We say that f has a $(t,\delta,\varepsilon,\lambda)$-*data structure* of length N if there exist an encoder $\mathcal{E} : D \to \{0,1\}^N$ and a (randomized) decoder $\mathcal{D}$ with the following properties: for every $x \in D$ and every $w \in \{0,1\}^N$ at Hamming distance $\Delta(w,\mathcal{E}(x)) \le \delta N$,

(1) $\mathcal{D}$ makes at most t bit-probes to w,
(2) $\Pr[\mathcal{D}^w(q) \in \{f(x,q),\perp\}] \ge 1-\varepsilon$ for every $q \in Q$,
(3) the set $G = \{q : \Pr[\mathcal{D}^w(q) = f(x,q)] \ge 1-\varepsilon\}$ has size at least $(1-\lambda)|Q|$ ('G' stands for "good"),
(4) if $w = \mathcal{E}(x)$, then $G = Q$.

Here $\mathcal{D}^w(q)$ denotes the random variable which is the decoder's output on inputs w and q. The notation indicates that it accesses the two inputs in different ways: while it has full access to the query q, it only has bit-probe access (or "oracle access") to the string w.

We say that a $(t,\delta,\varepsilon,\lambda)$-data structure is *error-correcting*, or an *error-correcting data structure*, if $\delta > 0$. Setting $\lambda = 0$ recovers the original notion of error-correction in de Wolf's model [19]. A $(t,\delta,\varepsilon,\lambda)$*-relaxed locally decodable code (RLDC)*, defined in [3], is an error-correcting data structure for the membership function $f : \{0,1\}^n \times [n] \to \{0,1\}$, where $f(x,i) = x_i$. A (t,δ,ε)-*locally decodable code (LDC)*, defined by Katz and Trevisan [14], is an RLDC with $\lambda = 0$.

Remark 1.2. For the data structure problems considered in this paper, our decoding procedures make only *non-adaptive* probes, i.e., the positions of the probes are determined all at once and sent simultaneously to the oracle. For other data structure problems it may be natural for decoding procedures to be adaptive. Thus, we do not require $\mathcal{D}$ to be non-adaptive in Condition 1 of Definition 1.1.

1.1. Our results

We obtain efficient error-correcting data structures for the following two data structure problems.

MEMBERSHIP: Consider a universe $[n] = \{1,\ldots,n\}$ and some nonnegative integer $s \leq n$. Given a set $S \subseteq [n]$ with at most s elements, one would like to store S in a compact representation that can answer "membership queries" efficiently, i.e., given an index $i \in [n]$, determine whether or not $i \in S$. Formally $D = \{S : S \subseteq [n], |S| \leq s\}$, $Q = [n]$, and $A = \{0,1\}$. The function $\textsc{Mem}_{n,s}(S,i)$ is 1 if $i \in S$ and 0 otherwise.

Since there are at least $\binom{n}{s}$ subsets of the universe of size at most s, each subset requiring a different instantiation of the data structure, the information-theoretic lower bound on the space of any data structure is at least $\log\binom{n}{s} \approx s\log n$ bits.[2] An easy way to achieve this is to store S in sorted order. If each number is stored in its own $\log n$-bit "cell," this data structure takes s cells, which is $s\log n$ bits. To answer a membership query, one can do a binary search on the list to determine whether $i \in S$ using about $\log s$ "cell-probes," or $\log s \cdot \log n$ bit-probes. The length of this data structure is essentially optimal, but its number of probes is not. Fredman, Komlós, and Szemerédi [11] developed a famous hashing-based data structure that has length $O(s)$ cells (which is $O(s\log n)$ bits) and only needs a *constant* number of cell-probes (which is $O(\log n)$ bit-probes). Buhrman, Miltersen, Radhakrishnan, and Venkatesh [5] improved upon this by designing a data structure of length $O(s\log n)$ bits that answers queries with *only one bit-probe* and a small error probability. This is simultaneously optimal in terms of time (clearly one bit-probe cannot be improved upon) and space (up to a constant factor).

None of the aforementioned data structures can tolerate a constant fraction of noise. To protect against noise for this problem, de Wolf [19] constructed an error-correcting data structure with $\lambda = 0$ using a locally decodable code (LDC). That construction answers membership queries in t bit-probes and has length roughly $L(s,t)\log n$, where $L(s,t)$ is the shortest length of an LDC encoding s bits with bit-probe complexity t. Currently, all known LDCs with $t = O(1)$ have $L(s,t)$ super-polynomial in s [2, 21, 7]. In fact, $L(s,t)$ must be super-linear for all constant t, see e.g. [14, 16, 20].

Under our present model of error-correction, we can construct much more efficient data structures with error-correcting capability. First, it is not hard to show that by composing the BMRV data structure [5] with the error-correcting data structure for $\textsc{Mem}_{n,n}$ (equivalently, an RLDC) [3], one

[2] Our logs are always to base 2.

can already obtain an error-correcting data structure of length $O((s \log n)^{1+\eta})$, where η is an arbitrarily small constant. However, following an approach taken in [19], we obtain a data structure of length $O(s^{1+\eta} \log n)$, which is much shorter than the aforementioned construction if $s = o(\log n)$.

Theorem 1.3. *For every $\varepsilon, \eta \in (0,1)$, there exist an integer $t > 0$ and real $\tau > 0$, such that for all s and n, and every $\delta \leq \tau$, $\textsc{Mem}_{n,s}$ has a $(t, \delta, \varepsilon, \frac{s}{2n})$-data structure of length $O(s^{1+\eta} \log n)$.*

We will prove Theorem 1.3 in Section 2. Note that the size of the good set G is at least $n - \frac{s}{2}$. Hence corrupting a δ-fraction of the bits of the data structure may cause a decoding failure for at most half of the queries $i \in S$ but not all. One may replace this factor $\frac{1}{2}$ easily by another constant (though the parameters t and τ will then change).

POLYNOMIAL EVALUATION: Let $\mathbb{Z}_n$ denote the set of integers modulo n and $s \leq n$ be some nonnegative integer. Given a univariate polynomial $g \in \mathbb{Z}_n[X]$ of degree at most s, we would like to store g in a compact representation so that for each evaluation query $a \in \mathbb{Z}_n$, $g(a)$ can be computed efficiently. Formally, $D = \{g : g \in \mathbb{Z}_n[X], \deg(g) \leq s\}$, $Q = \mathbb{Z}_n$, and $A = \mathbb{Z}_n$, and the function is $\textsc{PolyEval}_{n,s}(g, a) = g(a)$.

Since there are n^{s+1} polynomials of degree at most s, with each polynomial requiring a different instantiation of the data structure, the information-theoretic lower bound on the space of any data structure for this problem is at least $\log(n^{s+1}) \approx s \log n$ bits. Since each answer is an element of $\mathbb{Z}_n$ and must be represented by $\lfloor \log n \rfloor + 1$ bits, $\lfloor \log n \rfloor + 1$ is the information-theoretic lower bound on the bit-probe complexity.

Consider the following two naive solutions. On one hand, one can simply record the evaluations of g in a table with n entries, each with $\lfloor \log n \rfloor + 1$ bits. The length of this data structure is $O(n \log n)$ and each query requires reading only $\lfloor \log n \rfloor + 1$ bits. On the other hand, g can be stored as a table of its $s + 1$ coefficients. This gives a data structure of length and bit-probe complexity $(s + 1)(\lfloor \log n \rfloor + 1)$.

A natural question is whether one can construct a data structure that is optimal both in terms of space and time, i.e., has length $O(s \log n)$ and answers queries with $O(\log n)$ bit-probes. No such constructions are known to exist. However, some lower bounds are known in the weaker cell-probe model, where each cell is a sequence of $\lfloor \log n \rfloor + 1$ bits. For instance, as noted in [18], any data structure for POLYNOMIAL EVALUATION that stores $O(s^2)$ cells ($O(s^2 \log n)$ bits) requires reading at least $\Omega(s)$ cells. Moreover, by [17], if $\log n \gg s \log s$ and the data structure is constrained to store $s^{O(1)}$ cells, then its query complexity is $\Omega(s)$ cells. This implies that the second trivial construction described above is essentially optimal in the cell-probe model.

Recently, Kedlaya and Umans [15] obtained a data structure of length $s^{1+\eta} \log^{1+o(1)} n$ (where η is an arbitrarily small constant) that answers evaluation queries with $O(\operatorname{polylog} s \cdot \log^{1+o(1)} n)$ bit-probes. These parameters exhibit the best tradeoff between s and n so far. When $s = n^\eta$ for some $0 < \eta < 1$, the data structure of Kedlaya and Umans [15] is much superior to the trivial solution: its length is nearly optimal, and the query complexity drops from $\operatorname{poly} n$ to only $\operatorname{polylog} n$ bit-probes.

Here we construct an error-correcting data structure for the polynomial evaluation problem that works even in the presence of adversarial noise, with length nearly linear in $s \log n$ and bit-probe complexity $O(\operatorname{polylog} s \cdot \log^{1+o(1)} n)$. Formally:

Theorem 1.4. *For every $\varepsilon, \lambda, \eta \in (0,1)$, there exists $\tau \in (0,1)$ such that for all positive integers $s \leq n$, for all $\delta \leq \tau$, the data structure problem $\textsc{PolyEval}_{n,s}$ has a $(O(\operatorname{polylog} s \cdot \log^{1+o(1)} n), \delta, \varepsilon, \lambda)$-data structure of length $O((s \log n)^{1+\eta})$.*

Remark 1.5. We note that Theorem 1.4 easily holds when $s = (\log n)^{o(1)}$. As we discussed previously, one can just store a table of the $s+1$ coefficients of g. To make this error-correcting, encode the entire table by a standard error-correcting code. This has length and bit-probe complexity $O(s \log n) = O(\log^{1+o(1)} n)$.

1.2. Our techniques

At a high level, for both data structure problems we build our constructions by composing a relaxed locally decodable code with an appropriate noiseless data structure. If the underlying probe-accessing scheme in a noiseless data structure is "pseudorandom," then the noiseless data structure can be made error-correcting by appropriate compositions with other data structures. By pseudorandom, we mean that if a query is chosen uniformly at random from Q, then the positions of the probes selected also "behave" as if they are chosen uniformly at random. Such property allows us to analyze the error-tolerance of our constructions.

More specifically, for the MEMBERSHIP problem we build upon the noiseless data structure of Buhrman et al. [5]. While de Wolf [19] combined this with LDCs to get a rather long data structure with $\lambda = 0$, we will combine it here with RLDCs to get nearly optimal length with small (but non-zero) λ. In order to bound λ in our new construction, we make use of the fact that the [5]-construction is a bipartite *expander graph*, as explained below after Theorem 2.2. This property wasn't needed in [19]. The left side of the expander represents the set of queries, and a neighborhood of a query (a left node) represents the set of possible bit-probes that can be chosen to answer this query. The expansion property of the graph essentially implies that for a random query, the distribution of a bit-probe chosen to answer this query is close to uniform.[3] This property allows us to construct an efficient, error-correcting data structure for this problem.

For the polynomial evaluation problem, we rely upon the noiseless data structure of Kedlaya and Umans [15], which has a decoding procedure that uses the reconstruction algorithm from the Chinese Remainder Theorem. The property that we need is the simple fact that if a is chosen uniformly at random from $\mathbb{Z}_n$, then for any $m \leq n$, a modulo m is uniformly distributed in $\mathbb{Z}_m$. This implies that for a random evaluation point a, the distribution of certain tuples of cell-probes used to answer this evaluation point is close to uniform. This observation allows us to construct an efficient, error-correcting data structure for polynomial evaluation. Our construction follows the non-error-correcting one of [15] fairly closely; the main new ingredient is to add redundancy to their Chinese Remainder-based reconstruction by using more primes, which gives us the error-correcting features we need.

Time-complexity of decoding and encoding. So far we have used the number of bit-probes as a proxy for the actual time the decoder needs for query-answering. This is fairly standard, and usually justified by the fact that the actual time complexity of decoding is not much worse than its number of bit-probes. This is also the case for our constructions. For MEMBERSHIP, it can be shown that the decoder uses $O(1)$ probes and $\mathrm{polylog}(n)$ time (as do the RLDCs of [3]). For POLYNOMIAL EVALUATION, the decoder uses $\mathrm{polylog}(s) \log^{1+o(1)}(n)$ probes and $\mathrm{polylog}(sn)$ time.

The efficiency of *encoding*, i.e., the "pre-processing" of the data into the form of a data structure, for both our error-correcting data structures MEMBERSHIP and POLYNOMIAL EVALUATION

[3] We remark that this is different from the notion of smooth decoding in the LDC literature, which requires that for every *fixed* query, each bit-probe by itself is chosen with probability close to uniform (though not independent of the other bit-probes).

depends on the efficiency of encoding of the RLDC constructions in [3]. This is not addressed explicitly there, and needs further study.

2. The MEMBERSHIP problem

In this section we construct a data structure for the membership problem $\textsc{Mem}_{n,s}$. First we describe some of the building blocks that we need to prove Theorem 1.3. Our first basic building block is the relaxed locally decodable code of Ben-Sasson et al. [3] with nearly linear length. Using our terminology, we can restate their result as follows:

Theorem 2.1 (BGHSV [3]). *For every $\varepsilon \in (0, 1/2)$ and $\eta > 0$, there exist an integer $t > 0$ and reals $c > 0$ and $\tau > 0$, such that for every n and every $\delta \le \tau$, the membership problem $\textsc{Mem}_{n,n}$ has a $(t, \delta, \varepsilon, c\delta)$-data structure for $\textsc{Mem}_{n,n}$ of length $O(n^{1+\eta})$.*

Note that by picking the error-rate δ a sufficiently small constant, one can set $\lambda = c\delta$ (the fraction of unrecoverable queries) to be very close to 0.

The other building block that we need is the following one-probe data structure of Buhrman et al. [5].

Theorem 2.2 (BMRV [5]). *For every $\varepsilon \in (0, 1/2)$ and for every positive integers $s \le n$, there is an $(1, 0, \varepsilon, 0)$-data structure for $\textsc{Mem}_{n,s}$ of length $m = \frac{100}{\varepsilon^2} s \log n$ bits.*

Properties of the BMRV encoding: The encoding can be represented as a bipartite graph $\mathcal{G} = (L, R, E)$ with $|L| = n$ left vertices and $|R| = m$ right vertices, and regular left degree $d = \frac{\log n}{\varepsilon}$. This $\mathcal{G}$ is an *expander graph*: for each set $S \subseteq L$ with $|S| \le 2s$, its neighborhood $\Gamma(S)$ satisfies $|\Gamma(S)| \ge \left(1 - \frac{\varepsilon}{2}\right) |S|d$. For each assignment of bits to the left vertices with at most s ones, the encoding specifies an assignment of bits to the right vertices. In other words, each $x \in \{0, 1\}^n$ of weight $|x| \le s$ corresponds to an assignment to the left vertices, and the m-bit encoding of x corresponds to an assignment to the right vertices.

For each $i \in [n]$ we write $\Gamma_i := \Gamma(\{i\})$ to denote the set of d neighbors of i. A crucial property of the encoding function $\mathcal{E}_{bmrv}$ is that for every x of weight $|x| \le s$, for each $i \in [n]$, if $y = \mathcal{E}_{bmrv}(x) \in \{0, 1\}^m$ then $\Pr_{j\in\Gamma_i}[x_i = y_j] \ge 1 - \varepsilon$. Hence the decoder for this data structure can just probe a random index $j \in \Gamma_i$ and return the resulting bit y_j. Note that this construction is not error-correcting at all, since $|\Gamma_i|$ errors in the data structure suffice to erase all information about the i-th bit of the encoded x. ■

As we mentioned in the Section 1.1, by combining the BMRV encoding with the data structure for $\textsc{Mem}_{n,n}$ from Theorem 2.1, one easily obtains an $(O(1), \delta, \varepsilon, O(\delta))$-data structure for $\textsc{Mem}_{n,s}$ of length $O((s \log n)^{1+\eta})$. However, we can give an even more efficient, error-correcting data structure of length $O(s^{1+\eta} \log n)$. Our improvement follows an approach taken in de Wolf [19], which we now describe. For a vector $x \in \{0, 1\}^n$ with $|x| \le s$, consider a BMRV structure encoding $20n$ bits into m bits. The following "balls and bins estimate" is known:

Proposition 2.3 (From Section 2.3 of [19]). *For every positive integers $s \le n$, the BMRV bipartite graph $\mathcal{G} = ([20n], [m], E)$ for $\textsc{Mem}_{20n,s}$ with error parameter $\frac{1}{10}$ and $m = 10^4 s \log(20n)$ has the following property: there exists a partition of $[m]$ into $b = 10 \log(20n)$ disjoint sets $B_1, \ldots, B_b$ of $10^3 s$ vertices each, such that for each $i \in [n]$, there are at least $\frac{b}{4}$ sets B_k satisfying $|\Gamma_i \cap B_k| = 1$.*

Proposition 2.3 suggests the following encoding and decoding procedures. To encode x, we rearrange the m bits of $\mathcal{E}_{bmrv}(x)$ into $\Theta(\log n)$ disjoint blocks of $\Theta(s)$ bits each, according to the partition guaranteed by Proposition 2.3. Then for each block, encode these bits with the error-correcting data structure (RLDC) from Theorem 2.1. Given a received word w, to decode $i \in [n]$, pick a block B_k at random. With probability at least $\frac{1}{4}$, $\Gamma_i \cap B_k = \{j\}$ for some j. Run the RLDC decoder to decode the j-th bit of the k-th block of w. Since most blocks don't have much higher error-rate than the average (which is at most δ), with high probability we recover $\mathcal{E}_{bmrv}(x)_j$, which equals x_i with high probability. Finally, we can argue that most queries do not receive a blank symbol $\perp$ as an answer, using the expansion property of the BMRV encoding structure. Due to space limitation, we give only a proof sketch of Theorem 1.3 here.

Proof of Theorem 1.3. We only construct an error-correcting data structure with error probability 0.49. By a standard amplification technique we can reduce the error probability to any other positive constant (i.e., repeat the decoder $O(\log(1/\varepsilon))$ times).

By Theorem 2.2, there exists an encoder $\mathcal{E}_{bmrv}$ for an $(1, 0, \frac{1}{10}, 0)$-data structure for the membership problem $\textsc{Mem}_{20n,s}$ of length $m = 10^4 s \log(20n)$. Let $s' = 10^3 s$. By Theorem 2.1, for every $\eta > 0$, for some $t = O(1)$, and sufficiently small δ, $\textsc{Mem}_{s',s'}$ has a $(t, 10^5\delta, \frac{1}{100}, O(\delta))$-data structure of length $s'' = O(s'^{1+\eta})$. Let $\mathcal{E}_{bghsv}$ and $\mathcal{D}_{bghsv}$ be its encoder and decoder, respectively.

Encoding. Let $B_1, \dots, B_b$ be a partition of $[m]$ as guaranteed by Proposition 2.3. For a string $w \in \{0,1\}^m$, we abuse notation and write $w = w_{B_1} \cdots w_{B_b}$ to denote the string obtained from w by applying the permutation on $[m]$ according to the partition $B_1, \dots, B_b$. In other words, w_{B_k} is the concatenation of w_i where $i \in B_k$. We now describe the encoding process.

Encoder $\mathcal{E}$: on input $x \in \{0,1\}^n$, $|x| \leq s$,

(1) Let $y = \mathcal{E}_{bmrv}\left(x0^{19n}\right)$ and write $y = y_{B_1} \dots y_{B_b}$.
(2) Output the concatenation $\mathcal{E}(x) = \mathcal{E}_{bghsv}\left(y_{B_1}\right) \dots \mathcal{E}_{bghsv}\left(y_{B_b}\right)$.

The length of $\mathcal{E}(x)$ is $N = b \cdot O(s'^{1+\eta}) = O(s^{1+\eta} \log n)$.

Decoding. Given a string $w \in \{0,1\}^N$, we write $w = w^{(1)} \dots w^{(b)}$, where for $k \in [b]$, $w^{(k)}$ denotes the s''-bit string $w_{s''\cdot(k-1)+1} \dots w_{s''\cdot k}$.

Decoder $\mathcal{D}$: on input i and with oracle access to a string $w \in \{0,1\}^N$,

(1) Pick a random $k \in [b]$.
(2) If $|\Gamma_i \cap B_k| \neq 1$, then output a random bit.
Else, let $\Gamma_i \cap B_k = \{j\}$. Run and output the answer given by the decoder $\mathcal{D}_{bghsv}(j)$, with oracle access to the s''-bit string $w^{(k)}$.

Analysis. We defer the analysis to the full version [6]. ■

3. The POLYNOMIAL EVALUATION problem

In this section we prove Theorem 1.4. Given a polynomial g of degree s over $\mathbb{Z}_n$, our goal is to write down a data structure of length roughly linear in $s \log n$ so that for each $a \in \mathbb{Z}_n$, $g(a)$ can be computed with roughly $\operatorname{polylog} s \cdot \log n$ bit-probes. Our data structure is built on the work of Kedlaya and Umans [15]. Since we cannot quite use their construction as a black-box, we first give a high-level overview of our proof, motivating each of the proof ingredients that we need.

Encoding based on reduced polynomials: The most naive construction, by recording $g(a)$ for each $a \in \mathbb{Z}_n$, has length $n \log n$ and answers an evaluation query with $\log n$ bit-probes. As explained in [15], one can reduce the length by using the Chinese Remainder Theorem (CRT): If P_1 is a collection of distinct primes, then a nonnegative integer $m < \prod_{p \in P_1} p$ is uniquely specified by (and can be reconstructed efficiently from) the values $[m]_p$ for each $p \in P_1$, where $[m]_p$ denotes $m \mod p$.

Consider the value $g(a)$ over $\mathbb{Z}$, which can be bounded above by n^{s+2}, for $a \in \mathbb{Z}_n$. Let P_1 consist of the first $\log(n^{s+2})$ primes. For each $p \in P_1$, compute the reduced polynomial $g_p := g \mod p$ and write down $g_p(b)$ for each $b \in \mathbb{Z}_p$. Consider the data structure that simply concatenates the evaluation table of every reduced polynomial. This data structure has length $|P_1|(\max_{p \in P_1} p)^{1+o(1)}$, which is $s^{2+o(1)} \log^{2+o(1)} n$ by the Prime Number Theorem. Note that $g(a) < \prod_{p \in P_1} p$. So to compute $[g(a)]_n$, it suffices to apply CRT to reconstruct $g(a)$ over $\mathbb{Z}$ from the values $[g(a)]_p = g_p([a]_p)$ for each $p \in P_1$. The number of bit-probes is $|P_1| \log(\max_{p \in P_1} p)$, which is $s^{1+o(1)} \log^{1+o(1)} n$.

Error-correction with reduced polynomials: The above CRT-based construction has terrible parameters, but it serves as an important building block from which we can obtain a data structure with better parameters. For now, we explain how the above CRT-based encoding can be made error-correcting. One can protect the bits of the evaluation tables of each reduced polynomial by an RLDC as provided by Theorem 2.1. However, the evaluation tables can have non-binary alphabets, and a bit-flip in just one "entry" of an evaluation table can destroy the decoding process. To remedy this, one can first encode each entry by a standard error-correcting code and then encode the concatenation of all the tables by an RLDC. This is encapsulated in Lemma 3.1, which can be viewed as a version of Theorem 2.1 over non-binary alphabet. We defer this proof to the full version of this paper [6].

Lemma 3.1. *Let $f : D \times Q \to \{0,1\}^\ell$ be a data structure problem. For every $\varepsilon, \eta, \lambda \in (0,1)$, there exists $\tau \in (0,1)$ such that for every $\delta \leq \tau$, f has an $(O(\ell), \delta, \varepsilon, \lambda)$-data structure of length $O((\ell|Q|)^{1+\eta})$.*

To apply Lemma 3.1, let D be the set of degree-s polynomials over $\mathbb{Z}_n$, Q be the set of all evaluation points of all the reduced polynomials of g (each $q \in Q$ specified by a pair (a, p) of an evaluation point a and a prime modulus p), and the data structure problem f outputs evaluations of some reduced polynomial of g.

By itself, Lemma 3.1 cannot guarantee resilience against noise. In order to apply the CRT to reconstruct $g(a)$, all the values $\{[g(a)]_p : p \in P_1\}$ must be correct, which is not guaranteed by Lemma 3.1. To fix this, we add redundancy, taking a larger set of primes than necessary so that the reconstruction via CRT can be made error-correcting. Specifically, we apply a Chinese Remainder Code, or CRT code for short, to the encoding process.

Definition 3.2 (CRT code)**.** Let $p_1 < p_2 < \ldots < p_N$ be distinct primes, $K < N$, and $T = \prod_{i=1}^{K} p_i$. The *Chinese Remainder Code (CRT code)* with basis $p_1, \ldots, p_N$ and rate $\frac{K}{N}$ over message space $\mathbb{Z}_T$ encodes $m \in \mathbb{Z}_T$ as $\langle [m]_{p_1}, [m]_{p_2}, \ldots, [m]_{p_N} \rangle$.

Remark 3.3. By CRT, for distinct $m_1, m_2 \in \mathbb{Z}_T$, their encodings agree on at most $K - 1$ coordinates. Hence the Chinese Remainder Code with basis $p_1 < \ldots < p_N$ and rate $\frac{K}{N}$ has distance $N - K + 1$.

It is known that good families of CRT code exist and that unique decoding algorithms for CRT codes can correct up to almost half of the distance of the code (see e.g., [12]). The following statement can be easily derived from known facts, and we defer its proof to the full version [6].

Theorem 3.4. *For every positive integer T, there exists a set P consisting of distinct primes, with (1) $|P| = O(\log T)$, and (2) $\forall p \in P, \log T < p < 500 \log T$, such that a CRT code with basis P and message space $\mathbb{Z}_T$ has rate $\frac{1}{2}$, relative distance $\frac{1}{2}$, and can correct up to a $(\frac{1}{4} - O(\frac{1}{\log\log T}))$-fraction of errors.*

We apply Theorem 3.4 to a message space of size n^{s+2} to obtain a set of primes P_1 with the properties described above. Note that these primes are all within a constant factor of one another, and in particular, the evaluation table of each reduced polynomial has the same length, up to a constant factor. This fact and Lemma 3.1 will ensure that our CRT-based encoding is error-correcting.

Reducing the bit-probe complexity: We now explain how to reduce the bit-probe complexity of the CRT-based encoding, using an idea from [15]. Write $s = d^m$, where $d = \log^C s$, $m = \frac{\log s}{C \log\log s}$, and $C > 1$ is a sufficiently large constant. Consider the following multilinear extension map $\psi_{d,m} : \mathbb{Z}_n[X] \to \mathbb{Z}_n[X_0, \ldots, X_{m-1}]$ that sends a univariate polynomial of degree at most s to an m-variate polynomial of degree less than d in each variable. For every $i \in [s]$, write $i = \sum_{j=0}^{m-1} i_j d^j$ in base d. Define $\psi_{d,m}$ which sends X^i to $X_1^{i_0} \cdots X_m^{i_{m-1}}$ and extends multilinearly to $\mathbb{Z}_n[X]$.

To simplify our notation, we write $\tilde{g}$ to denote the multivariate polynomial $\psi_{d,m}(g)$. For every $a \in \mathbb{Z}_n$, define $\tilde{a} \in \mathbb{Z}_n^m$ to be $([a]_n, [a^d]_n, [a^{d^2}]_n, \ldots, [a^{d^{m-1}}]_n)$. Note that for every $a \in \mathbb{Z}_n$, $g(a) = \tilde{g}(\tilde{a})$ (mod n). Now the trick is to observe that the total degree of the multilinear polynomial $\tilde{g}$ is less than the degree of the univariate polynomial g, and hence its maximal value over the integers is much reduced. In particular, for every $a \in \mathbb{Z}_n^m$, the value $\psi_{d,m}(g)(a)$ over the integers is bounded above by $d^m n^{dm+1}$.

We now work with the reduced polynomials of $\tilde{g}$ for our encoding. Let P_1 be the collection of primes guaranteed by Theorem 3.4 when $T_1 = d^m n^{dm+1}$. For $p \in P_1$, let $\tilde{g}_p$ denote $\tilde{g} \mod p$ and $\tilde{a}_p$ denote the point $([a]_p, [a^d]_p, \ldots, [a^{d^{m-1}}]_p)$. Consider the data structure that concatenates the evaluation table of $\tilde{g}_p$ for each $p \in P_1$. For each $a \in \mathbb{Z}_n$, to compute $g(a)$, it suffices to compute $\tilde{g}(\tilde{a})$ over $\mathbb{Z}$, which by Theorem 3.4 can be reconstructed (even with noise) from the set $\{\tilde{g}_p(\tilde{a}_p) : p \in P_1\}$.

Since the maximum value of $\tilde{g}$ is at most $T_1 = d^m n^{dm+1}$ (whereas the maximum value of g is at most $d^m n^{d^m+1}$), the number of primes we now use is significantly less. This effectively reduces the bit-probe complexity. In particular, each evaluation query can be answered with $|P_1| \cdot \max_{p \in P_1} \log p = (dm \log n)^{1+o(1)}$ bit-probes, which by our choice of d and m is equal to $\operatorname{polylog} s \cdot \log^{1+o(1)} n$. However, the *length* of this encoding is still far from the information-theoretically optimal $s \log n$ bits. We shall explain how to reduce the length, but since encoding with multilinear reduced polynomials introduces potential complications in error-correction, we first explain how to circumvent these complications.

Error-correction with reduced multivariate polynomials: There are two complications that arise from encoding with reduced multivariate polynomials. The first is that not all the points in the evaluation tables are used in the reconstructive CRT algorithm. Lemma 3.1 only guarantees that most of the entries of the table are decoded correctly with high probability, but not all of them (even if the fraction of errors in the table is low, a λ-fraction of queries may be answered by $\perp$). So if the entries that are used in the reconstruction via CRT are not decoded by Lemma 3.1, then the whole decoding procedure fails.

More specifically, to reconstruct $\tilde{g}(\tilde{a})$ over $\mathbb{Z}_n$, it suffices to query the point $\tilde{a}_p$ in the evaluation table of $\tilde{g}_p$ for each $p \in P_1$. Typically the set $\{\tilde{a}_p : a \in \mathbb{Z}_n\}$ will be much smaller than $\mathbb{Z}_p^m$, so not all the points in $\mathbb{Z}_p^m$ are used. To circumvent this issue, we only store the query points that are used in the CRT reconstruction. Let $B^p = \{\tilde{a}_p : a \in \mathbb{Z}_n\}$. For each $p \in P_1$, the encoding only stores the evaluation of $\tilde{g}_p$ at the points B^p instead of the entire domain $\mathbb{Z}_p^m$. The disadvantage of computing the evaluation at the points in B^p is that the encoding stage takes time proportional to n. We thus give up on encoding efficiency (which was one of the main goals of Kedlaya and Umans) in order to guarantee error-correction.

The second complication is that the sizes of the evaluation tables may no longer be within a constant factor of each other. (This is true even if the evaluation points come from all of $\mathbb{Z}_p^m$.) If one of the tables has length significantly longer than the others, then a constant fraction of noise may completely corrupt the entries of all the other small tables, rendering decoding via CRT impossible. This potential problem is easy to fix; we apply a repetition code to each evaluation table so that all the tables have equal length.

Reducing the length: Now we explain how to reduce the length of the data structure to nearly $s \log n$, along the lines of Kedlaya and Umans [15]. To reduce the length, we need to reduce the magnitude of the primes used by the CRT reconstruction. We can effectively achieve that by applying the CRT twice. Instead of storing the evaluation table of $\tilde{g}_p$, we apply CRT again and store evaluation tables of the reduced polynomials of $\tilde{g}_p$ instead. Whenever an entry of $\tilde{g}_p$ is needed, we can apply the CRT reconstruction to the reduced polynomials of $\tilde{g}_p$.

Note that for $p_1 \in P_1$, the maximum value of $\tilde{g}_{p_1}$ (over the integers rather than mod n) is at most $T_2 = d^m p_1^{dm+1}$. Now apply Theorem 3.4 with T_2 the size of the message space to obtain a collection of primes P_2. Recall that each $p_1 \in P_1$ is at most $O(dm \log n)$. So each $p_2 \in P_2$ is at most $O((dm)^{1+o(1)} \log\log n)$, which also bounds the cardinality of P_2 from above.

For each query, the number of bit-probes made is at most $|P_1||P_2| \max_{p_2 \in P_2} \log p_2$, which is at most $(dm)^{2+o(1)} \log^{1+o(1)} n$. Recall that by our choice of d and m, $dm = \frac{\log^{C+1} s}{C \log\log s}$. Thus, the bit-probe complexity is $\mathrm{polylog}\, s \cdot \log^{1+o(1)} n$. Now, by Lemma 3.1, the length of the encoding is nearly linear in $|P_1||P_2| \max_{p_2 \in P_2} p_2^m \log p_2$, which is at most $\mathrm{polylog}\, s \cdot \log^{1+o(1)} n \cdot \max_{p_2 \in P_2} p_2^m$. So it suffices to bound $\max_{p_2 \in P_2} p_2^m$ from above. To this end, recall that by the remark following Theorem 1.4, we may assume without loss of generality that $s = \Omega(\log^{\zeta} n)$ for some $0 < \zeta < 1$. This implies that $\log\log\log n \leq \log\log s - \log \zeta$. Then for each $p_2 \in P_2$,

$$\begin{aligned} p_2^m &\leq \left(O\left((dm)^{1+o(1)} \log\log n\right)\right)^m \\ &\leq (dm)^{(1+o(1))m} \cdot s^{\frac{1}{C}+o(1)}. \end{aligned}$$

It is easy to see that $(dm)^{(1+o(1))m}$ can be bounded above by $s^{(1+o(1))(1+\frac{1}{C}-o(1))}$. Thus, $p_2^m = s^{1+\frac{2}{C}+o(1)}$. Putting everything together, the length of the encoding is nearly linear in $s \log n$. As mentioned, we defer the formal proof to the full version of this paper [6].

Acknowledgments

We thank Madhu Sudan for helpful comments and suggestions on the presentation of this paper.

References

[1] Y. Aumann and M. Bender. Fault-tolerant data structures. In *Proceedings of 37th IEEE FOCS*, pages 580–589, 1996.

[2] A. Beimel, Y. Ishai, E. Kushilevitz, and J. Raymond. Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic Private Information Retrieval. In *Proceedings of 43rd IEEE FOCS*, pages 261–270, 2002.

[3] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006. Earlier version in STOC'04.

[4] G. Brodal, R. Fagerberg, I. Finocchi, F. Grandoni, G. Italiano, A. Jørgenson, G. Moruz, and T. Mølhave. Optimal resilient dynamic dictionaries. In *Proceedings of 15th European Symposium on Algorithms (ESA)*, pages 347–358, 2007.

[5] H. Buhrman, P. B. Miltersen, J. Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? *SIAM Journal on Computing*, 31(6):1723–1744, 2002. Earlier version in STOC'00.

[6] V. Chen, E. Grigorescu, and R. de Wolf. Efficient and Error-Correcting Data Structures for Membership and Polynomial Evaluation, 2009. Preprint at http://arxiv.org/abs/0909.3696.

[7] K. Efremenko. 3-query locally decodable codes of subexponential length. In *Proceedings of 41st ACM STOC*, 2009.

[8] I. Finocchi, F. Grandoni, and G. Italiano. Optimal resilient sorting and searching in the presence of memory faults. In *Proceedings of 33rd ICALP*, volume 4051 of *Lecture Notes in Computer Science*, pages 286–298, 2006.

[9] I. Finocchi, F. Grandoni, and G. Italiano. Resilient search trees. In *Proceedings of 18th ACM-SIAM SODA*, pages 547–553, 2007.

[10] I. Finocchi and G. Italiano. Sorting and searching in the presence of memory faults (without redundancy). In *Proceedings of 36th ACM STOC*, pages 101–110, 2004.

[11] M. Fredman, M. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, 1984.

[12] O. Goldreich, D. Ron, and M. Sudan. Chinese remaindering with errors. *IEEE Transactions on Information Theory*, 46(4):1330–1338, 2000.

[13] A. G. Jørgenson, G. Moruz, and T. Mølhave. Resilient priority queues. In *Proceedings of 10th International Workshop on Algorithms and Data Structures (WADS)*, volume 4619 of *Lecture Notes in Computer Science*, 2007.

[14] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of 32nd ACM STOC*, pages 80–86, 2000.

[15] K. S. Kedlaya and C. Umans. Fast modular composition in any characteristic. In *Proceedings of 49th IEEE FOCS*, pages 146–155, 2008.

[16] I. Kerenidis and R. de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *Journal of Computer and System Sciences*, 69(3):395–420, 2004. Earlier version in STOC'03. quant-ph/0208062.

[17] P. B. Miltersen. On the cell probe complexity of polynomial evaluation. *Theor. Comput. Sci.*, 143(1):167–174, 1995.

[18] P. B. Miltersen. Cell probe complexity - a survey. Invited paper at *Advances in Data Structures* workshop. Available at Miltersen's homepage, 1999.

[19] R. de Wolf. Error-correcting data structures. In *Proceedings of 26th Annual Symposium on Theoretical Aspects of Computer Science (STACS'2009)*, pages 313–324, 2009. cs.DS/0802.1471.

[20] D. Woodruff. New lower bounds for general locally decodable codes. Technical report, ECCC Report TR07–006, 2006.

[21] S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. *Journal of the ACM*, 55(1), 2008. Earlier version in STOC'07.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 215-226
www.stacs-conf.org

LOG-SPACE ALGORITHMS FOR PATHS AND MATCHINGS IN k-TREES

BIRESWAR DAS [1] AND SAMIR DATTA [2] AND PRAJAKTA NIMBHORKAR [1]

[1] The Institute of Mathematical Sciences
Chennai, India
E-mail address: {bireswar,prajakta}@imsc.res.in

[2] Chennai Mathematical Institute, Chennai, India
E-mail address: sdatta@cmi.ac.in

Abstract. Reachability and shortest path problems are NL-complete for general graphs. They are known to be in L for graphs of tree-width 2 [14]. However, for graphs of tree-width larger than 2, no bound better than NL is known. In this paper, we improve these bounds for k-trees, where k is a constant. In particular, the main results of our paper are log-space algorithms for reachability in directed k-trees, and for computation of shortest and longest paths in directed acyclic k-trees.

Besides the path problems mentioned above, we consider the problem of deciding whether a k-tree has a perfect macthing (decision version), and if so, finding a perfect matching (search version), and prove that these problems are L-complete. These problems are known to be in P and in RNC for general graphs, and in SPL for planar bipartite graphs [8].

Our results settle the complexity of these problems for the class of k-trees. The results are also applicable for bounded tree-width graphs, when a tree-decomposition is given as input. The technique central to our algorithms is a careful implementation of divide-and-conquer approach in log-space, along with some ideas from [14] and [19].

1. Introduction

Reingold's striking result [21], showed that undirected reachability is in L, thus collapsing the class SL to L. On the other hand, directed reachability, which happens to be NL-complete is another similar sounding problem for which there is only partial progress to report. A result of Allender and Reinhardt, [22] hints at a partial collapse of NL by showing that directed reachability is in the formally smaller class UL, although, *non-uniformly.*

In the absence of better constructive upper bounds it is natural to consider natural restrictions on graphs which allow us to improve the upper bounds on reachability and related problems. Typical examples of this approach are [1],[23], where the complexity of various versions of planar and somewhat non-planar (in the sense of excluding only a K_5 or only a $K_{3,3}$ minor) are considered. In the same spirit, but using different techniques, [14]

1998 ACM Subject Classification: Computational Complexity.
Key words and phrases: k-trees, reachability, matching, log-space.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2456

considers reachability and related questions in series-parallel graphs and places all of these in L. They leave open the question of complexity of such problems in bounded tree-width graphs. Series-parallel graphs have tree-width two and happen to be planar. But higher tree widths graphs are highly non-planar. In fact, any k-tree for $k > 4$ contains both K_5 and $K_{3,3}$.

We resolve the open questions posed in [14] and show a matching L lower bound to complete the characterization of reachability problems in k-trees. Thus one of the main results of our paper is the following:

Theorem 1.1. *The following problems are L-complete:*

1. *Computing reachability between two vertices in directed k-trees,*
2. *Computing shortest and longest paths in directed acyclic k-trees.*

In this paper, we also consider the perfect matching problem. The parallel complexity of perfect matching problems is a long standing open problem where the best known algorithms use randomness as a resource [20],[15]. Even in the planar case, the search problem for perfect matchings is known to be in NC for bipartite graphs only [8].

We prove a complete characterization for the decision and search versions of the perfect matching problem for k-trees. This improves significantly upon previous best known upper bound of LogCFL for bounded tree-width graphs. Thus another main result of our paper is:

Theorem 1.2. *Deciding whether a k-tree has a perfect matching, and if so, finding a perfect matching is L-complete.*

Our primary technique is a careful use of divide-and-conquer to enable the algorithm to run in L. However, for the distance computation we need to import a constructive version of tree separation from [19] where it is stated in the context of Visibly Pushdown Automata (VPAs). We believe that porting this technique for use in general log-space computation is an important contribution of this paper.

At this point, we must mention an important caveat. All our log-space results hold directly only for k-trees and not for partial k-trees which are also equivalent to tree-width k graphs. The reason being that a tree decomposition for partial k-trees is apparently more difficult to construct (best known upper bound is LogCFL[24]) as opposed to k-trees (for which it can be done in L [17]). Having mentioned that it is important to observe that if we are given the tree decomposition of a partial k-tree, we can do the rest of computation in L.

The rest of the paper is organized as follows: Section 2 gives the necessary background. Section 3 contains log-space algorithms for reachability in directed k-paths and k-trees. Section 4 contains log-space algorithms for shortest and longest path in directed acyclic k-paths and k-trees. Section 5 contains log-space algorithms for perfect matching problems in a k-tree.

2. Preliminaries

We define k-trees and a subclass of k-trees known as k-paths here, and also describe a suitable representation for the graphs in these two classes. This representation is used in our algorithms in the rest of the paper. All the definitions given here are applicable to both directed as well as undirected graphs. For directed graphs, the directions of the edges can be ignored while defining k-trees and k-paths and while computing their suitable representations.

The class of graphs known as k-trees is defined as (cf. [12]):

Definition 2.1. The class of *k-trees* is inductively defined as follows.

- A clique with k vertices (k-clique for short) is a k-tree.
- Given a k-tree G' with n vertices, a k-tree G with $n+1$ vertices can be constructed by picking a k-clique X (called the *support*)in G' and then joining a new vertex v to each vertex u in X. Thus, $V(G) = V(G') \cup \{v\}$, $E(G) = E(G') \cup \{\{u, v\} \mid u \in X\}$.

A *partial k-tree* is a subgraph of a k-tree. The class of partial k-trees coincides with the class of graphs that have tree-width at most k. k-trees are recognizable in log-space [2] but partial k-trees are not known to be recognizable in log-space. In literature, several different representations of k-trees have been considered [10, 2, 17]. We use the following representation given by Köbler and Kuhnert [17]:

Definition 2.2. Let $G = (V, E)$ be a k-tree. The tree representation $T(G)$ of G is defined by

$$V(T(G)) = \{M \subseteq V \mid M \text{ is a } k\text{-clique or a } (k+1)\text{-clique}\},$$
$$E(T(G)) = \{\{M_1, M_2\} \subseteq V \mid M_1 \subsetneq M_2\}$$

In [17], it is proved that $T(G)$ is a tree and can be computed in log-space. In the rest of the paper, we use G in place of $T(G)$. Thus, by *a k-tree G*, we always mean that G is in fact represented as $T(G)$. The term *vertices in G* refers to the vertices in the original graph, whereas *a node in G* and *a clique in G* refer to the nodes of $T(G)$. Partial k-trees also have a tree-decomposition similar to that of k-trees, which is also not known to be log-space computable.

k-paths is a sub-class of k-trees (e.g. see [11]). The recursive definition of k-paths is similar to that of k-trees. However, a new vertex can be added only to a particular clique called the *current clique*. After addition of a vertex, the current clique may remain the same, or may change by dropping a vertex and adding the new vertex in the current clique. We consider the following representation of k-paths, which is based on the recursive definition of k-paths, and is known to be computable in log-space [2]:

Given a k-path $G = (V, E)$, for $i = 1, \cdots, m$, let X_i be the current cliques at the ith stage of the recursive construction of the k-path. Let $V_1 = \cup_i X_i$ and $V_2 = V \setminus V_1$. We call the vertices in V_2 as *spikes*. The following facts are easy to see:

1. No two spikes have an edge between them.
2. Each spike is connected to all the vertices of exactly one of the X_i's.
3. X_i and X_{i+1} share exactly $k-1$ vertices

The representation of G consists of a graph $G' = (V', E')$ where $V' = \{X_1, \ldots, X_m\} \cup V_2$ and $E' = \{(X_i, X_{i+1}) \mid 1 \leq i < m\} \cup \{(X, v) | X \text{ is a clique in } \in V', v \in V_2 \text{ has a neighbour in } X\}$.

3. Reachability

We give log-space algorithms to compute reachability in k-paths and in k-trees. Although the graphs considered in this section are directed, when we refer to any of the definitions or decompositions in Section 2, we consider the underlying undirected graph.

3.1. Reachability in k-paths

Without loss of generality, we can assume that s and t are vertices in some k-cliques X_i and X_j, and not spikes. If s (t) is a spike, then it has at most k out-neighbors (resp. in-neighbors) and we can take one of the out-neighbors (resp. in-neighbors) as the new source s' and new sink t' and check reachability. As there are only k^2 such pairs, we can cycle through all of them in log-space. The algorithm is based on the observation that a simple s to t path ρ can pass through any clique at most k times. We use a divide- and-conquer approach similar to that used in Savitch's algorithm (which shows that directed reachability can be computed in $DSPACE(\log^2 n)$). The main steps involved in the algorithm are as follows:

1. Preprocessing step: Make the cliques disjoint by labeling different copies of each vertex with different labels and introducing appropriate edges. Compute reachabilities within each clique including its spikes, and *remove the spikes*. Number the cliques $X_1, \ldots, X_m$ left to right.

2. Now assume that s and t are in cliques X_i and X_j respectively. Note that $i = j$ is also possible, but without loss of generality, we can assume $i < j$. This is because, if $i = j$, we can make another copy X_i' of X_i, join the copies of the same vertex by bidirectional edges to preserve reachabilities, and choose the copy of s from X_i and that of t from X_i'.

3. Divide the k-path into three parts P_1, P_2 and P_3 where P_1 consists of cliques $X_1, \ldots, X_i$, P_2 consists of $X_i, \ldots, X_j$, and P_3 consists of $X_j, \ldots, X_m$. Note that X_i (X_j) appears in both P_1 and P_2 (P_2 and P_3 respectively). Now compute reachabilities of all pairs of vertices in X_i (X_j) when the graph is restricted to P_1 (respectively P_3). Then the reachability of t from s within P_2 is computed, using the previously computed reachabilities within P_1 and P_3.

Each of these steps can be done by a log-space transducer. The details are given below.

Preprocessing: Although adjacent k-cliques in a k-path decomposition share $k-1$ vertices, we perform a preprocessing step, where we give distinct labels to each copy of a vertex. As all the copies of a vertex form a (connected) sub-path in the k-path decomposition, we join two copies of a vertex appearing in two adjacent cliques by bidirectional edges. It can be seen that this preserves reachabilities. Any copy of s and t can be taken as the new s and t. Another preprocessing step involves removing the spikes maintaining reachabilities between all pairs of vertices, and computing reachabilities within each k-clique. Both of these preprocessing steps can be done by a log-space transducer. The proof appears in the full version of the paper.

The Algorithm: We describe an algorithm to compute pairwise reachabilities in X_i and X_j in P_1 and P_3 respectively, and also s-t reachability in P_2 using these previously computed pairwise reachabilities. Algorithm 1 describes this reachability routine. The routine gets as input two vertices u and v, and two indices i and j. It determines whether v is reachable from u in the sub-path $P = (X_i, \ldots, X_j)$. This input is given in such a way that u and v always lie in X_i or X_j. Consider the case when both u and v are in X_i (or both in X_j). Let l be the center of P. Then a path from u to v either lies entirely in the sub-path $P' = (X_i, \ldots, X_l)$ or it crosses X_l at most k times. Thus if $X_l = \{v_1, \ldots, v_k\}$ then for $\{v_{i_1}, \cdots, v_{i_r}\} \subseteq X_l$ we need to check reachabilities between u and say v_{i_1} in P', then between v_{i_1} and v_{i_2} in $P'' = (X_l, \ldots, X_j)$ and so on, and finally between v_{i_r} and v in P'. It suffices to check all the r-tuples in X_l, where $0 \leq r \leq k$. The case when $u \in X_i$ and $v \in X_j$ (and vice versa) is analogous. In Algorithm 1, we present only one case where

$u, v \in X_i$. Other three cases are analogous. Thus at each recursive call, the length of the sub-path under consideration is halved, and $O(\log m)$ iterations suffice. The algorithm can be implemented in log-space. The correctness and complexity analysis of the algorithm appears in the full version.

Algorithm 1 Procedure IsReach(u, v, i, j)

```
Input: Pre-processed k-path decomposition of graph $G$, clique indices $i, j$, vertex labels
  $u, v \in X_i$. {Other three cases are analogous.}
Decide: Whether $v$ is reachable from $u$ in sub-path $P = (X_i, \dots, X_j)$.
if $j - i = 1$ then
  Compute the reachability directly, as the sub-path has only $2k$ vertices.
  Return the result.
end if
$l = \frac{j+i}{2}$
if $u, v \in X_i$ then
  if IsReach($u$, $v$, $i$, $l$) then
    Return 1;
  else
    for $q = 1$ to $k$ do
      $v_0 \leftarrow u$, $v_{q+1} \leftarrow v$
      for all $q$-tuples $(v_1, \dots, v_q)$ of vertices in $X_l$ do
        if $\bigwedge_{\substack{x=0 \\ x \text{ even}}}^{q+1}$ IsReach($v_x$,$v_{x+1}$,$i$,$l$) $\wedge$ $\bigwedge_{\substack{x=1 \\ x \text{ odd}}}^{q+1}$ IsReach($v_x$,$v_{x+1}$,$l$,$j$) then
          Return 1;
        end if
      end for
    end for
  end if
end if
```

3.2. Reachability in k-trees

Given a directed k-tree G in its tree decomposition and two vertices s and t in G, we describe a log-space algorithm that checks whether t is reachable from s. This algorithm uses Algorithm 1 as a subroutine and involves the following steps: The complexity analysis is given in Lemma 3.1.

1. **Preprocessing:** Like k-paths, assign distinct labels to the copies of each vertex u in different cliques. Introduce a bidirectional edge between the copies of u in all the adjacent pairs of cliques. As reachabilities are maintained during this process, any copy of s and t can be taken as the new s and t. Let X_i and X_j be the cliques containing s and t respectively.

2. **The Procedure:** After this preprocessing, we have a tree T with its nodes as disjoint k-cliques of vertices of G, and s and t are contained in cliques X_i and X_j. Compute the unique undirected path ρ between X_i and X_j in T in log-space. Each node on ρ has two of its neighbors on ρ, except X_i and X_j, which have one neighbor each. An s to t path has to cross each clique in ρ, and additionally, it can pass through the subtrees attached to

each node X_l on ρ. Hence for each node X_l on ρ, we pre-compute the pairwise reachabilities among the k vertices contained in X_l when the k-tree is restricted to the subtree rooted at X_l. We define the subtree rooted at X_l as the subtree consisting of X_l and those nodes which can be reached from X_l without going through any node on ρ. Note that once this is done for each node X_l on ρ, we are left with ρ. As ρ is a k-path, we can use Algorithm 1 in Section 3.1 to compute reachabilities within ρ.

3. **Computing reachabilities within the subtree rooted at X_l:** We do this inductively. If the subtree rooted at X_l contains only one node X_l, we have only k vertices, and their pairwise reachabilities within X_l can be computed in $O(k \log k)$ space. We recursively find the reachabilities within the subtrees rooted at each of the children of X_l. Let the size of the subtree rooted at X_l be N. At most one of the children of X_l can have a subtree of size larger than $\frac{N}{2}$. Let X_a be such a child. Recursively compute the pairwise reachabilities for each pair of vertices in X_a within the subtree rooted at X_a. The reachabilities are represented as a $k \times k$ boolean matrix referred to as the *reachability matrix* M for the vertices in X_a, when the graph is confined to the subtree rooted at X_a. M is then used to compute the pairwise reachabilities of vertices in X_l, when the graph is confined to X_l and the subtree rooted at X_a. This gives a new matrix M' of size k^2. It is stored on stack while computing the reachability matrix M'' for another child X_b of X_l. The matrix M' is updated using M'', so that it represents reachabilities between each pair of vertices in X_l when the graph is confined to X_l and the subtrees rooted at X_a and X_b. This process is continued till all the children of X_l are processed. The matrix M' at this stage reflects the pairwise reachabilities between vertices of X_l, when the graph is confined to the subtree rooted at X_l. Note that the storage required while making a recursive call is only the current reachability matrix M'. Recall that M' contains the pairwise reachabilitities among the vertices in X_l in the subgraph corresponding to X_l and the subtrees rooted at those children of X_l which are processed so far. We give the complexity analysis in the full version.

Lemma 3.1. *The procedure described above can be implemented in log-space.*

Hardness for L: L-hardness of reachability in k-trees follows from L-hardness of the problem of path ordering (proved to be SL-hard in [9], and is L-hard due to SL=L result of [21]). We give the details in the full version.

4. Shortest and Longest Paths

We show that the shortest and longest paths in weighted directed acyclic k-trees can be computed in log-space, when the weights are positive and are given in unary. Throughout this section, the terms k-path and k-tree always refer to directed acyclic k-paths and k-trees respectively, with integer weights on edges and we here onwards omit the specification *weighted directed acyclic*. We use the following (weighted) form of the result from [18]: The proof is exactly similar to that in [18] and we omit it here.

Theorem 4.1 (See[18], Theorem 9)**.** *Let $\mathcal{C}$ be any subclass of weighted directed acyclic graphs closed under vertex deletions. There is a function f, computable in log-space with oracle access to* Reach($\mathcal{C}$)*, that reduces* Distance($\mathcal{C}$) *to* Long-Path($\mathcal{C}$) *and* Long-Path($\mathcal{C}$) *to* Distance($\mathcal{C}$)*, where* Reach($\mathcal{C}$), Distance($\mathcal{C}$)*, and* Long-Path($\mathcal{C}$) *are the problems of deciding reachability, computing distance and longest path respectively for graphs in $\mathcal{C}$.*

We use this theorem to reduce the shortest path problem in k-trees to the longest path problem, and then compute the longest (that is, maximum weight) s to t path. The reduction involves changing the weights of the edges such that the shortest path becomes the longest path and vice versa. This gives a directed acyclic k-tree with positive integer weights on edges given in unary. The class of k-trees is not closed under vertex deletions. However, once a tree decomposition of a k-tree is computed, deleting vertices from the cliques leaves some cliques of size smaller than k, which does not affect the working of the algorithm.

We show that the maximum weight of an s to t path can be computed in log-space using a technique which uses ideas from [14]. The algorithm to compute maximum weight s to t path in k-trees uses the algorithm for computing maximum weight path in k-paths as subroutine. Therefore we first describe the algorithm for k-paths in Section 4.1

4.1. Maximum Weight Path in Directed Acyclic k-paths

Let G be a directed acyclic k-path and s and t be two designated vertices in G. The computation of maximum weight of an s to t path is done in five stages, described below in detail. The main idea is to obtain a log-depth circuit by a suitable modification of Algorithm 1, and to transform this circuit to an arithmetic formula over integers, whose value is used to compute the maximum weight of an s to t path in G.

Computing the maximum weight s to t path in G involves the following steps:

(1) **Construct a log-depth formula from Algorithm 1:** Modify Algorithm 1 so that it outputs a circuit $\mathcal{C}$ that has nodes corresponding to the recursive calls made in Line 15 and the tuples considered in the **for** loop in Line 14. A node q in $\mathcal{C}$ that corresponds to a recursive call IsReach(u, v, i, j) has children $q_1, \cdots, q_N$, which correspond to the tuples considered in that recursive call (**for**-loop on Line 12 of Algorithm 1). We refer to q as a *call-node* and $q_1, \ldots, q_N$ as *tuple-nodes.* A tuple-node q' corresponding to a tuple $(v_1, \ldots, v_N)$ has call-nodes $q'_1, \ldots, q'_N$ as its children, which correspond to the recursive calls made while considering the tuple $(v_1, \ldots, v_N)$ (Line 15 of Algorithm 1). The leaves of $\mathcal{C}$ are those recursive calls which satisfy the **if** condition on Line 3 of Algorithm 1, thus they are always call-nodes. As the depth of the recursion in Algorithm 1 is $O(\log n)$, the circuit $\mathcal{C}$ also has $O(\log n)$ depth. Hence it can be converted to a formula $\mathcal{F}$ by only a polynomial factor blow-up in its size. The maximum number of children of a node is $O(k^k)$ and hence the size of $\mathcal{F}$ is bounded by $O(k^{k \log n})$, which is polynomial in n for constant k.

(2) **Prune the boolean formula:** The internal call-nodes of $\mathcal{F}$ are replaced by $\vee$ gates and tuple-nodes are replaced by $\wedge$ gates. The leaves of $\mathcal{F}$ are replaced by 0 or 1 depending on whether the corresponding recursive call returned 0 or 1 in the **if** block on Line 3 of Algorithm 1. It can be seen that a sub-formula of $\mathcal{F}$ rooted at a call-node evaluates to 1 if and only if the corresponding recursive call returns 1 in Algorithm 1. Similarly, the sub-formula rooted at a tuple-node evaluates to 1 if and only if the conjunction corresponding to it (on Line 15 of Algorithm 1) evaluates to 1. Now, we evaluate the sub-formula rooted at each node of $\mathcal{F}$. Note that a node that evaluates to 0 does not contribute to any path from s to t, and hence its subtree can be safely removed.

(3) **Transformation into a $\{+, max\}$-tree:** The new, pruned formula obtained in Step 2 is then relabeled: Each $\wedge$ label is replaced with a $+$ label and each $\vee$ label with a *max* label. Each leaf corresponds to calls of the form $IsReach(u, v, i, i + 1)$. It is labeled with the length of the maximum weight u to v path confined within cliques i and $i + 1$, which can be computed in $O(1)$ space. This weight is strictly positive, since the 0-weight leaves are removed in Step 2. Further, all the weights are in unary. Thus we now have a $\{+, max\}$-tree T with positive, unary weights on its leaves. It is easy to see that the value of the $\{+, max\}$-tree T is the maximum weight of any s to t path in G.

(4) **Transformation into a $\{+, \times\}$-tree:** The evaluation problem on the $\{+, max\}$-tree T obtained in Step 3 is then reduced to the evaluation problem on a $\{+, \times\}$-tree T' whose leaves are labeled with positive integer weights coded in binary. This reduction works in log-space and is similar to that of [14]. The reduction involves replacing a $+$-node of T with a $\times$-node, and a *max*-node with a $+$ node. The weight w of a leaf is replaced with r^{mw}, where r is the smallest power of 2 such that $r \geq n$, and m is the sum of the weights of all the leaves of T plus one. The correctness of the reduction follows from a similar result in [14], and we omit the proof here.

(5) **Evaluation of the $\{+, \times\}$ tree:** This can be done in log-space due to [5, 3, 7, 13]. The value of T is $v = \lfloor \frac{log_r v'}{m} \rfloor$.

4.2. Maximum Weight Path in Directed Acyclic k-trees

Given a directed acyclic k-tree (in its tree-decomposition) G, two vertices s and t in G, and weights on the edges of G, encoded in unary, we show how to compute the maximum weight of an s to t path in G. Unlike the case of k-paths, the reachability algorithm for k-trees given in Section 3.2 can not be used to get a log-depth circuit since the recursion depth of the algorithm is same as the depth of the k-tree. Therefore we need to find another way of recursively dividing the k-tree into smaller and smaller subtrees, as we did for k-paths in Sections 3.1 and 4.1. This is based on the technique used in the following result of [19]:

Lemma 4.2. *(***Lemma** 6 **of** [19]**, also see** [4]*) Let M be a visibly pushdown automaton accepting well-matched strings over an alphabet Δ. Given an input string x, checking whether $x \in L(M)$ can be done in log-space.*

Using Lemma 4.2, we can compute *a set of recursive separators for a tree* defined below:

Definition 4.3. Given a rooted tree T, separators of T are two nodes a and b of T such that

1. The subtrees rooted at a and b respectively are disjoint,

2. T is split into subtrees T_1, T_2, T_3 where T_1 consists of a, some (or possibly all) of the children of a, and subtrees rooted at them, T_2 is defined similarly for b, and T_3 consists of the rest of the tree along with a copy of a and b each.

3. Each of T_1, T_2, T_3 consists of at most a $\frac{3}{4}$ fraction of the leaves of T.

This process is done recursively for T_1, T_2, T_3, until the number of leaves in the subtrees is two. Such a subtree is in fact a path. A *set of recursive separators of* T consists of the separators of T and of all the subtrees obtained in the recursive process.

The following lemma gives the procedure to compute a set of recursive separators of a tree T:

Lemma 4.4. *Given a tree T, the set of recursive separators of T can be computed in log-space.*

Proof. The algorithm of [19] deals with well-matched strings. An example of a well-matched string is a balanced parentheses expression, which is a string over $\{(,)\}$. In [19], a log-space algorithm is given for membership testing in those languages which are subsets of well-matched strings and are accepted by visibly pushdown automata. We restrict ourselves to balanced parentheses expressions. To check whether a string on parentheses is in the language, the algorithm of [19] recursively partitions the string into three disjoint substrings, such that each of the parts forms a balanced parentheses expression, and length of each part is at most $\frac{3}{4}$th of the length of the original string. To use this algorithm, we order the children of each node of T in a specific way, label the leaves with parentheses ‘($'$ and ‘)$'$ such that the leaves of the subtree rooted at any internal node form a string on balanced parentheses. We add dummy leaves if needed. The steps are as follows:

1. By adding dummy leaves, ensure that each internal node has an even number of children which are leaves, and there are at least two such children.

2. Arrange the children of each node from left to right such that the non-leaves are consecutive, and they have an equal number of leaves to the left and to the right.

3. For each internal node, label the left half of its leaf-children with ‘(’ and the right ones by ‘)’. This ensures that the leaves of the subtree rooted at each internal node form a balanced parentheses expression. Conversely, leaves which form a balanced parentheses expression are consecutive leaves in the subtree rooted at an internal node.

The leaves of T now form a balanced parentheses expression, and we run the algorithm of [19] on this string. The recursive splitting of the string into smaller substrings corresponds to the recursive splitting of T at some internal nodes, which satisfies Definition 4.3. This is ensured by the way the leaves are labeled. Each balanced parentheses expression corresponds to either a subtree rooted at an internal node or the subtrees rooted at some of the children of an internal node.

The subtrees obtained by splitting a tree have at most $\frac{3}{4}$th of the number of leaves in the tree. Thus at each stage of recursion, the number of leaves in the subtrees is reduced by a constant fraction. Moreover, the algorithm of [19] can output all the substrings formed at each stage of recursion in log-space. As a substring completely specifies a subtree of T, our procedure outputs the set of recursive separators for T in log-space. ■

Once an algorithm to compute the set of recursive separators for k-trees is known, a reachability routine similar to Algorithm 1 can be designed in a straight forward way. We give the details in the full version. From the reachability routine, the computation of maximum weight path follows from the steps 1 to 5 described in Section 4.1.

4.3. Distance Computation in Undirected k-trees

We give a simple log-space algorithm for computing the shortest path between two given vertices in an undirected k-tree. We use the decomposition of [16], where a k-tree is decomposed into layers. We use the following properties of the decomposition:

1. Layer 0 is a k-clique. Each vertex in layer $i > 0$ has exactly k neighbors in layers $j < i$. Further, these neighbors of i which are in layers lower than that of i form a k-clique.

2. No two vertices in the same layer share an edge.

This decomposition is log-space computable [17]. Moreover, given two vertices s and t, it is always possible to find a decomposition in which t lies in layer 0. This can also be done in log-space. If both s and t are in layer 0, then there is an edge between s and t, which is the shortest path from s to t. Therefore assume that s lies in a layer $r > 0$. The following claim leads to a simple algorithm. The proof appears in the full version.

Claim 4.5. 1. The shortest s to t path never passes through two vertices u and v such that $layer(u) < layer(v)$. 2. There is a shortest path from s to t passing through the neighbor of s in the lowest layer.

This claim suggests a simple algorithm which can be implemented in log-space: Start from s and choose the next vertex from the lowest possible layer, at each step till we reach layer 0.

5. Perfect Matching in k-trees

Hardness for L: To show that the decision version of perfect matching is hard for L, we show that the problem of path ordering, can be reduced to the perfect matching problem for k-trees. We give the proof in the full version:

Lemma 5.1. *Determining whether a k-tree has a perfect matching is L-hard.*

L upper bounds: We describe a log-space algorithm to decide whether a k-tree has a perfect matching and, if so, output a perfect matching. The algorithm is inspired by an $O(n^3)$ algorithm [6] for computing the matching polynomial in series-parallel graphs. The idea is to exploit the fact that k-trees have a tree decomposition of bounded width, so that any perfect matching of the entire k-tree induces a partial matching on any subtree which leaves at most constantly many vertices unmatched. Thus we generalize the problem to that of determining, for each set, S, of constantly many vertices in the root of the subtree, whether there is a matching of the subtree that leaves exactly the vertices in S unmatched. Now we "recursively" solve the generalized problem and for this purpose we need to maintain a bit-vector indexed by the sets S which is still of bounded length. The algorithm composes the bit-vectors of the children of a node to yield the bit-vector for the node. The bit-vector, which we refer to as *matching vector*, is defined as follows:

Definition 5.2. Let G be a k-tree with tree-decomposition T. T has alternate levels of k-cliques and $k+1$-cliques. Root T arbitrarily at a k-clique. Let s be a node in T that shares vertices $\{u_1, \ldots, u_k\}$ with its parent. Further, let H be the subgraph of G corresponding to the subtree of T rooted at s. The *matching vector* for s is a vector $\vec{v}_H = (v_H^{(S_1)}, \ldots, v_H^{(S_{2^k})})$ of dimension 2^k, where $S_1, \ldots, S_{2^k}$ are all the distinct subsets of $\{u_1, \ldots, u_k\}$, and $v_H^{(S_i)} = 1$ if H has a matching in which all the vertices of H matched, except those in S_i, $v_H^{(S_i)} = 0$ if there is no such matching.

It can be seen that G has a perfect matching if and only if $v_G^{(\emptyset)} = 1$.We show how to compute $\vec{v}_G$ in L, and also show how to construct a perfect matching in G, if one exists. We prove Part 1 of the following theorem. For a proof of part 2, we refer to the full version.

Theorem 5.3. *1. The problem of deciding whether a k-tree has a perfect matching is in* L.
2. Finding a perfect matchings in a k-tree is in FL.

Proof. (**of** 1) We compute the matching vector for the root by recursively computing the matching vectors of each of its children. For a leaf node in the tree-decomposition, the matching vector can be computed in a brute-force way. At an internal node s, the matching vector is computed from the matching vectors of its children, which we describe here:

Case 1: s is a k-node Let s has vertices $V_s = \{u_1, \ldots, u_k\}$. Recall that a k-node shares all its vertices with all its neighbors. Let the children of s in T be $s_1, \ldots, s_r$. Let the subgraph corresponding to the subtree rooted at s be H and those at its children be $H_1, \ldots, H_r$. In order to determine $v_H^{(S)}$, we need to know if there is a matching in H that leaves exactly the vertices in S unmatched. This holds if and only if the vertices in S are not matched in any of the H_j's, and each vertex in $V_s \setminus S$ is matched in exactly one of the H_j's. In other words, we need to determine if there is a partition $T_1, T_2, ..., T_r$ of $V_s \setminus S$, such that H_j has a matching in which precisely $V_s \setminus T_j$ is unmatched. That is, $v_{H_j}^{(V_s \setminus T_j)} = 1$ for all $1 \leq j \leq r$. More formally,

$$v_H^{(S)} = \bigvee_{\substack{T_1,\ldots,T_r \subseteq V_s \setminus S: \\ \forall j \neq j' \in [r] T_j \cap T_{j'} = \emptyset: \\ \cup_{j \in [r]} T_j = V_s \setminus S}} \bigwedge_{j \in [r]} v_{H_j}^{(V_s \setminus T_j)} = \bigvee_{\emptyset = U_0 \subseteq \ldots \subseteq U_r = V_s \setminus S} \bigwedge_{j \in [r]} v_{H_j}^{(V_s \setminus (U_j \setminus U_{j-1}))} \qquad (5.1)$$

where, the second equality follows by defining $U_0 = \emptyset$ and $U_i = \cup_{j \in [i]} T_j$ for $i \in [r]$. The size of the above DNF formula depends on r which is not a constant hence the straightforward implementation of the above computation would not be in L. However, consider a conjunct in the big disjunction in the second line above. The j^{th} factor of this conjunct depends only on U_j and U_{j-1}, each of which can be represented by a constant number ($= 2^k$) of bits. Thus, we can iteratively extend U_{j-1} in all possible ways to U_j and use the bit indexed by $V_s \setminus (U_j \setminus U_{j-1})$ in the vector for the child. How to obtain the vector of the child within a log-space bound is detailed in the full version.

Case 2: s is a $k+1$ node The procedure is slightly more complex in this case. Let s have vertices $\{u_1, \ldots, u_{k+1}\}$. Let the subgraph corresponding to the subtree rooted at s be H. Let $s_1, \ldots, s_r$ be the children of s, with corresponding subgraphs $H_1, \ldots, H_r$. Note that s may share a different subset of k vertices with each of its children and with its parent. Let the vertices s shares with its parent be $\{u_1, \ldots, u_k\}$. Then its matching vector is indexed by the subsets of $\{u_1, \ldots, u_k\}$, and moreover, u_{k+1} should always be matched in H. To compute $\vec{v}_H$, we first extend the matching vectors of each of its children and make a 2^{k+1} dimensional vector $\vec{w}_H$. The matching vector $\vec{v}_{H_j}$ of a child s_j of s is extended to the new vector $\vec{w}_{H_j}$ as follows: Let s_j contain $\{u_1, \ldots, u_k\}$. We consider an entry $v_{H_j}^{(S)}$ of $\vec{v}_{H_j}$. The vector $\vec{w}_{H_j}$ has two entries corresponding to it.

$$w_{H_j}^{(S \cup \{u_{k+1}\})} = v_{H_j}^{(S)}, \qquad w_{H_j}^{(S)} = \bigvee_{\substack{p \in [k], u_p \notin S, \\ (u_{k+1}, v_p) \in E}} u_{H_j}^{(S \cup \{u_p\})}$$

These new vectors of each of the children can be composed similar to that in the previous case to get $\vec{w}_H$. To get $\vec{v}_H$, we remove the 2^k entries from $\vec{w}_H$ which are indexed on subsets containing u_{k+1}. This vector is passed on to the parent of s. The complexity analysis, and a proof of (2) appears in the full version. ∎

References

[1] Eric Allender, David Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. Planar and grid graph reachability problems. *Theory of Computing Systems*, 45, 2009.

[2] V. Arvind, B. Das, and J. Köbler. The Space Complexity of k-Tree Isomorphism. In *In Proceedings of ISAAC*, 2007.

[3] Michael Ben-or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.

[4] Burchard von Braunmühl and Rutger Verbeek. Input driven languages are recognized in log n space. In *Selected papers of the international conference on "foundations of computation theory" on Topics in the theory of computation*, pages 1–19, 1985.

[5] S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.*, 21(4):755–780, 1992.

[6] N. Chandrasekharan and S. Hannenhalli. Efficient algorithms for computing matching and chromatic polynomials on series-parallel graphs. *Computing and Information Proceedings, (ICCI 92)*, 1992.

[7] A. Chiu, G. Davida, and B. Litow. Division in logspace-uniform NC^1. *Theoretical Informatics and Applications*, 35, 2001.

[8] Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. In *STACS 2008*, volume 1 of *Leibniz International Proceedings in Informatics*, 2008.

[9] Kousha Etessami. Counting quantifiers, successor relations, and logarithmic space. *J. Comput. Syst. Sci.*, 54(3):400–411, 1997.

[10] J. G. Del Greco, C. N. Sekharan, and R. Sridhar. Fast parallel reordering and isomorphism testing of k-trees. *Algorithmica*, 32(1):61–72, 2002.

[11] A. Gupta, N. Nishimura, A. Proskurowski, and P. Ragde. Embeddings of k -connected graphs of pathwidth k. *Discrete Applied Mathematics*, 145(2):242–265, 2005.

[12] F. Harary and E. M. Palmer. On acyclic simplicial complexes. *Mathematica*, 15, 1968.

[13] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *JCSS*, 65(4), 2002.

[14] Andreas Jakoby and Till Tantau. Logspace algorithms for computing shortest and longest paths in series-parallel graphs. In *Proceedings of 27th FSTTCS, LNCS 4855*, 2007.

[15] Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

[16] M. M. Klawe, D. G. Corneil, and A. Proskurowski. Isomorphism testing in hookup classes. *SIAM Journal on Algebraic and Discrete Methods*, 3(2):260–274, 1982.

[17] Johannes Köbler and Sebastian Kuhnert. The isomorphism problem for $\boldsymbol{k}$-trees is complete for logspace. *ECCC*, (TR09-053), 2009.

[18] Nutan Limaye, Meena Mahajan, and Prajakta Nimbhorkar. Longest paths in planar dags in unambiguous log-space. In *Computing: Australasian Theory Symposium (CATS)*, 2009.

[19] Nutan Limaye, Meena Mahajan, and B. V. Raghavendra Rao. Arithmetizing classes around $NC^{\mathbf{1}}$ and L. In *STACS*, 2007.

[20] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[21] Omer Reingold. Undirected $\boldsymbol{st}$-connectivity in logspace. In *Proc. 37th STOC*, 2005.

[22] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. In *IEEE Symposium on Foundations of Computer Science*, pages 244–253, 1997.

[23] Thomas Thierauf and Fabian Wagner. Reachability in $\boldsymbol{K_{3,3}}$-free graphs and $\boldsymbol{K_5}$-free graphs is in unambiguous log-space. In *FCT*, 2009.

[24] Egon Wanke. Bounded tree-width and LOGCFL. *J. Algorithms*, 16(3):470–491, 1994.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 227-238
www.stacs-conf.org

RESTRICTED SPACE ALGORITHMS FOR ISOMORPHISM ON BOUNDED TREEWIDTH GRAPHS

BIRESWAR DAS [1] AND JACOBO TORÁN [2] AND FABIAN WAGNER [3]

[1] Institute of Mathematical Sciences, Chennai, India
E-mail address: `bireswar@imsc.res.in`

[2] Institut für Theoretische Informatik, Universität Ulm, 89069 Ulm, Germany
E-mail address: `jacobo.toran@uni-ulm.de`

[3] Institut für Theoretische Informatik, Universität Ulm, 89069 Ulm, Germany
E-mail address: `fabian.wagner@uni-ulm.de`

Abstract. The Graph Isomorphism problem restricted to graphs of bounded treewidth or bounded tree distance width are known to be solvable in polynomial time [2],[19]. We give restricted space algorithms for these problems proving the following results:

- Isomorphism for bounded tree distance width graphs is in L and thus complete for the class. We also show that for this kind of graphs a canon can be computed within logspace.
- For bounded treewidth graphs, when both input graphs are given together with a tree decomposition, the problem of whether there is an isomorphism which respects the decompositions (i.e. considering only isomorphisms mapping bags in one decomposition blockwise onto bags in the other decomposition) is in L.
- For bounded treewidth graphs, when one of the input graphs is given with a tree decomposition the isomorphism problem is in LogCFL.
- As a corollary the isomorphism problem for bounded treewidth graphs is in LogCFL. This improves the known TC^1 upper bound for the problem given by Grohe and Verbitsky [8].

1. Introduction

The Graph Isomorphism problem consists in deciding whether two given graphs are isomorphic, or in other words, whether there exists a bijection between the vertices of both graphs preserving the edge relation. Graph Isomorphism is a well studied problem in NP because of its many applications and also because it is one of the few natural problems in this class not known to be solvable in polynomial time nor known to be NP-complete. Although for the case of general graphs no efficient algorithm for the problem is known, the situation is much better when certain parameters in the input graphs are bounded by a constant. For

1998 ACM Subject Classification: Complexity Theory, Graph Algorithms.
Key words and phrases: Complexity, Algorithms, Graph Isomorphism Problem, Treewidth, LogCFL.
Supported by DFG grants TO 200/2-2.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2457

example the isomorphism problem for graphs of bounded degree [13], bounded genus [15], bounded color classes [14], or bounded treewidth [2] is known to be in P. Recently some of these upper bounds have been improved with the development of space efficient techniques, most notably Reingold's deterministic logspace algorithm for connectivity in undirected graphs [16]. In some cases logspace algorithms have been obtained. For example graph isomorphism for trees [12], planar graphs [5] or k-trees [10]. In other cases the problem has been classified in some other small complexity classes below P. The isomorphism problem for graphs of bounded treewidth is known to be in TC^1 [8] and the problem restricted to graphs of bounded color classes is known to be in the #L hierarchy [1].

In this paper we address the question of whether the isomorphism problem restricted to graphs of bounded treewidth and bounded tree distance width can be solved in logspace. Intuitively speaking, the treewidth of a graph measures how much it differs from a tree. This concept has been used very successfully in algorithmics and fixed-parameter tractability (see e.g. [3, 4]). For many complex problems, efficient algorithms have been found for the cases when the input structures have bounded treewidth. As mentioned above Bodlaender showed in [2] that Graph Isomorphism can be solved in polynomial time when restricted to graphs of bounded treewidth. More recently Grohe and Verbitsky [8] improved this upper bound to TC^1. In this paper we improve this result showing that the isomorphism problem for bounded treewidth graphs lies in LogCFL, the class of problems logarithmic space reducible to a context free language. LogCFL can be alternatively characterized as the class of problems computable by a uniform family of polynomial size and logarithmic depth circuits with bounded AND and unbounded OR gates, and is therefore a subclass of TC^1. LogCFL is also the best known upper bound for computing a tree decomposition of bounded treewidth graphs [18, 7], which is one bottleneck in our isomorphism algorithm. We prove that if tree decompositions of both graphs are given as part of the input, the question of whether there is an isomorphism respecting the vertex partition defined by the decompositions can be solved in logarithmic space. Our proof techniques are based on methods from recent isomorphism results [5, 6] and are very different from those in [8].

The notion of tree distance width, a stronger version of the treewidth concept, was introduced in [19]. There it is shown that for graphs with bounded tree distance width the isomorphism problem is fixed parameter tractable, something that is not known to hold for the more general class of bounded treewidth graphs. We prove that for graphs of bounded tree distance width it is possible to obtain a tree distance decomposition within logspace. Using this result we show that graph isomorphism for bounded tree distance width graphs can also be solved in logarithmic space. Since it is known that the question is also hard for the class L under AC^0 reductions [9], this exactly characterizes the complexity of the problem. We show that in fact a canon for graphs of bounded tree distance width, i.e. a fixed representative of the isomorphism equivalence class, can be computed in logspace. Due to space reasons, some proofs are omitted and will be provided in the full version of the paper.

2. Preliminaries

We introduce the complexity classes used in this paper. L is the class of decision problems computable by deterministic logarithmic space Turing machines. LogCFL consists of all decision problems that can be Turing reduced in logarithmic space to a context free language. There are several alternative more intuitive characterizations of LogCFL. Problems in this class can be computed by uniform families of polynomial size and logarithmic depth circuits over bounded fan-in AND gates and unbounded fan-in OR gates. We will also use the characterization of LogCFL as the class of decisional problems computable by non-deterministic auxiliary pushdown machines (NAuxPDA). These are Turing machines with a logarithmic space work tape, an additional pushdown and a polynomial time bound [17]. The class TC^1 contains the problems computable by uniform families of polynomial size and logarithmic depth threshold circuits. The known relationships among these classes are:

$$\mathrm{L} \subseteq \mathrm{LogCFL} \subseteq \mathrm{TC}^1.$$

In this paper we consider undirected simple graphs with no self loops. For a graph $G = (V, E)$ and two vertices $u, v \in V$, $d_G(u, v)$ denotes the distance between u and v in G (number of edges in the shortest path between u and v in G). For a set $S \subseteq V$, and a vertex $u \in V$, $d_G(S, u)$ denotes $\min_{v \in S} d_G(v, u)$. $\Gamma(S)$ denotes the set of neighbors of S in G. In a connected graph G, a separating set is a set of vertices such that deleting the vertices in S (and the edges connected to them) produces more than one connected component. For $G = (V, E)$ and two disjoint subsets U, W of V we use the following notion for an *induced bipartite subgraph* $B_G[U, W]$ of G on vertex set $U \cup W$ with edge set $\{\{u, w\} \in E \mid u \in U, w \in W\}$. Let $G[U]$ be the *induced subgraph* of G on vertex set $V \setminus U$.

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a collection of subsets of V called bags, and T is a tree with node set I and edge set F, satisfying the following properties:

i) $\bigcup_{i \in I} X_i = V$

ii) for each $\{u, v\} \in E$, there is an $i \in I$ with $u, v \in X_i$ and

iii) for each $v \in V$, the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of T.

The *width* of a tree decomposition of G, is defined as $\max\{|X_i| \mid i \in I\} - 1$. The *treewidth* of G is the minimum width over all tree decompositions of G.

A *tree distance decomposition* of a graph $G = (V, E)$ is a triple $(\{X_i \mid i \in I\}, T = (I, F), r)$, where $\{X_i \mid i \in I\}$ is a collection of subsets of V called *bags*, $X_r = S$ a set of vertices and T is a tree with node set I, edge set F and root r, satisfying:

i) $\bigcup_{i \in I} X_i = V$ and for all $i \neq j, X_i \cap X_j = \emptyset$

ii) for each $v \in V$, if $v \in X_i$ then $d_G(X_r, v) = d_T(r, i)$ and

iii) for each $\{u, v\} \in E(G)$, there are $i, j \in I$ with $u \in X_i, v \in X_j$ and $i = j$ or $\{i, j\} \in F$ (for every edge in G its two endpoints belong to the same or to adjacent bags in T).

Let $D = (\{X_i \mid i \in I\}, T = (I, F), r)$ be a tree distance decomposition of G. X_r is the *root bag* of D. The *width* of D is the maximum number of elements of a bag X_i. The *tree distance width* of G is the minimum width over all tree distance decompositions of G.

The tree distance decomposition D is called *minimal* if for each $i \in I$, the set of vertices in the bags with labels in the subtree rooted at i in T induce a connected subgraph in G. In [19] it is shown that for every root set $S \subseteq V$ there is a unique minimal tree distance decomposition of G with root set S. The width of such a decomposition is minimal among the tree distance decompositions of G with root set S.

An isomorphism from G onto H *respects* their tree (distance) decompositions D, D' if vertices in a bag of D in G are mapped blockwise onto vertices in a bag of D' in H. Not every isomorphism has this property.

$Sym(V)$ is the *symmetric group* on a set V.

3. Graphs of bounded tree distance width

3.1. Tree distance decomposition in L

We describe an algorithm that on input a graph G and a subset $S \subset V$ produces the minimal tree distance decomposition $D = (\{X_i \mid i \in I\}, T = (I, F), r)$ of G with root set $X_r = S$. The algorithm works within space $c \cdot k \log n$ for some constant c, where k is the width of the minimal tree distance decomposition of G with root set S. The output of the algorithm is a sequence of strings of the form (bag label, bag depth, $v_{i_1}, v_{i_2}, \ldots, v_{i_l}$), indicating the number of the bag, the distance of its elements to S and the list of the elements in the bag.

The algorithm basically performs a depth first traversal of the tree T in the decomposition while constructing it. Starting at S the algorithm uses three functions for traversing T. These functions perform queries to a logspace subroutine computing reachability [16].

Parent(X_i): On input the elements of a bag X_i the function returns the elements of the parent bag in T. These are the vertices $v \in V$ with the following two properties: $v \in \Gamma(X_i) \setminus X_i$ and v is reachable from S in $G \setminus X_i$. For a vertex v these two properties can be tested in space $O(\log n)$ by an algorithm with input G, S and X_i. In order to find all the vertices in the parent set, the algorithm searches through all the vertices in V.

First Child(X_i): This function returns the elements of the first child of i in T. This is the child with the vertex $v_j \in V$ with the smallest index j. v_j satisfies that $v_j \in \Gamma(X_i) \setminus X_i$ and that v_j is not reachable from S in $G \setminus X_i$. It can be found cycling in order through the vertices of G until the first one satisfying the properties is found. The other elements $w \in X_i$ must satisfy the same two properties as v_j and additionally, they must be in the same connected component in $G \setminus X_i$ where v_j is contained. In case X_i does not have any children, the function outputs some special symbol.

Next Sibling(X_i): This function first computes X_p :=Parent(X_i) and then searches for the child of p in T next to X_i. Let v_i be the vertex with the smallest label in X_i. This is done similarly as the computation of First Child. The next sibling is the bag containing the unique vertex v_j with the following properties: v_j is the vertex with the smallest label in this bag, $label(v_j) > label(v_i)$ and there is no other bag which has a vertex with a label $> v_i$ and $< v_j$. The vertex v_j is not reachable from S in $G \setminus X_p$. The other elements in the bag are the vertices satisfying these properties and which are in the same connected component of $G \setminus X_p$ where v_j is contained.

With these three functions the algorithm performs a depth-first traversal of T. It only needs to remember the initial bag $X_0 = S$ which is part of the input, and the elements of the current bag. On a bag X_i it searches for its first child. If it does not exist then it searches for the next sibling. When there are no further siblings the next move goes up in the tree T. The algorithm finishes when it returns to S. It also keeps two counters in order to be able to output the number and depth of the bags. The three mentioned functions only need to keep at most two bags (X_i and its father) in memory, and work in logarithmic space. On input a graph G with n vertices, and a root set S, the space used by the algorithm is therfore bounded by $c \cdot k \log n$, for a constant c, and k being the minimum width of a tree distance decomposition of G with root set S. When considering how the three functions are defined it is clear that the algorithm constructs a tree distance decomposition with root set S. Also they make sure that for each i the subgraph induced by the vertices of the bags in the subtree rooted at i is connected thus producing a minimal decomposition. As observed in [19], this is the unique minimal tree distance decomposition of G with root set S.

3.2. Isomorphism Algorithm for Bounded Tree Distance Width Graphs

For our isomorphism algorithm we use a tree called the *augmented tree* which is based on the underlying tree of a minimal tree distance decomposition. This augmented tree, apart from the bags, contains information about the separating sets which separate bags.

Definition 3.1. Let G be a bounded tree distance width graph with a minimal tree distance decomposition $D = (\{X_i \mid i \in I\}, T = (I, F), r)$. The *augmented tree* $\mathcal{T}_{(G,D)} = (I_{(G,D)}, F_{(G,D)}, r)$ corresponding to G and D is a tree defined as follows:

- The set of nodes of $\mathcal{T}_{(G,D)}$ is $I_{(G,D)}$ which contains two kinds of nodes, namely $I_{(G,D)} = I \cup J$. Those in I form the set of *bag nodes* in D, and those in J the *separating set nodes*. For each bag node $a \in I$ and each child b of a in T we consider the set $X_a \cap \Gamma(X_b)$, i.e. the *minimum separating set* in X_a which separates X_b from the root bag X_r in G. Let $M_{s_1^a}, \dots, M_{s_{l(a)}^a}$ be the set of all minimum separating sets in X_a, free of duplicates. There are nodes for these sets $s_1^a, \dots, s_{l(a)}^a$, the separating set nodes. We define $J = \bigcup_{a \in I}\{s_1^a, \dots, s_{l(a)}^a\}$. The node $r \in I$ is the root in $\mathcal{T}_{(G,D)}$.
- In $F_{(G,D)}$ there are edges between bag nodes $a \in I$ and the separating set nodes $s_1^a, \dots, s_{l(a)}^a \in J$ (edges between bag nodes and their children in the augmented tree). There are also edges between nodes $b \in I$ and s_j^a if $M_{s_j^a}$ is the minimum separating set in X_a which separates X_b from X_r (edges between bag nodes and their parents).

To simplify notation, we later say for example that $s_1, \dots, s_l$ are the children of a bag node a if the context is clear. The odd levels of the augmented tree T' correspond to bag nodes and the even levels correspond to separating set nodes.

Observe that for each node in the augmented tree, we associate a bag to a bag node and a minimum separating set to a separating set node. Hence, every vertex v in the original graph occurs in at least one associated component and it might occur in more than one, e.g. if v is contained in a bag and in a minimum separating set.

Let $T_{(G,D)}$ be an augmented tree of some minimal tree distance decomposition D of a graph G. Let a be a node of $T_{(G,D)}$. The subtree of $T_{(G,D)}$ rooted at a is denoted by T_a. Note that $T_{(G,D)} = T_r$ where X_r is the bag corresponding to the root of the tree distance decomposition D. We define $\mathsf{graph}(T_a)$ as the subgraph of G induced by all the vertices associated to at least one of the nodes of T_a. The *size* of T_a, denoted $|T_a|$ is the number of vertices which occur in at least one component which is associated to a node in T_a. Note, $|T_a|$ is polynomially related to $|\mathsf{graph}(T_a)|$, i.e. the number of vertices in the corresponding subgraph of G.

When given a tree distance decomposition the augmented tree can be computed in logspace. Using the result in Section 3.1 we immediately get:

Lemma 3.2. *Let G be a graph of bounded tree distance width. The augmented tree for G can be computed in logspace.*

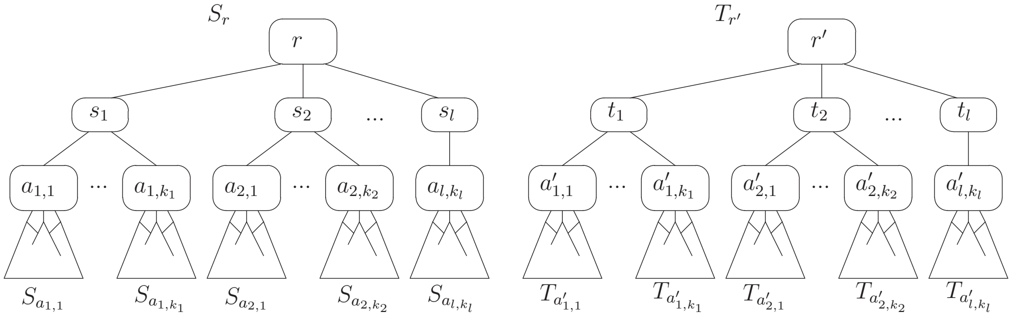


Figure 1: The augmented trees S_r and $T_{r'}$ rooted at bag nodes r and r'. Node r has separating set nodes $s_1, \ldots, s_l$ as children. The children of s_1 are again bag nodes $a_{1,1}, \ldots, a_{1,k_1}$. $S_{a_{i,j}}$ is the subtree rooted at $a_{i,j}$. Bag nodes and separating set nodes alternate in the tree.

Isomorphism Order of Augmented Trees. We describe an isomorphism order procedure for comparing two augmented trees $S_{(G,D)}$ and $T_{(H,D')}$ corresponding to the graphs G and H and their tree distance decompositions D and D', respectively. This isomorphism order algorithm is an extension of the one for trees given by Lindell [12] and it is different from that for planar graphs given by Datta et.al. [5]. The trees $S_{(G,D)}$ and $T_{(H,D')}$ are rooted at bag nodes r and r'. The rooted trees are denoted then S_r and $T_{r'}$ as shown in Figure 1.

We will show that two graphs of bounded tree distance width are isomorphic if and only if for some root nodes r and r' the augmented trees corresponding to the minimal tree distance decompositions have the same isomorphism order.

The isomorphism order depends on the order of the vertices in the bags r and r'. Let X_r and $X'_{r'}$ be the corresponding bags in D and D'. We define the sets of mappings $\Theta_{(r,r')} = Sym(X_r) \times Sym(X'_{r'})$. Let (σ, σ') be such a mapping, then the tuples $(G[X_r], \sigma)$ and $(G[X'_{r'}], \sigma')$ describe a fixed ordering on the vertices of the induced subgraphs. If r is not the top-level root of the augmented tree then $\Theta_{(r,r')}$ may become restricted to a subset, when going into recursion. The isomorphism order is defined to be $S_r <_{\mathtt{T}} T_{r'}$ if there exist mappings $(\sigma, \sigma') \in \Theta_{(r,r')}$ such that one of the following holds:

1) $(G[X_r], \sigma) < (H[X'_{r'}], \sigma')$ via lexicographical comparison of both ordered subgraphs
2) $(G[X_r], \sigma) = (H[X'_{r'}], \sigma')$ but $|S_r| < |T_{r'}|$
3) $(G[X_r], \sigma) = (H[X'_{r'}], \sigma')$ and $|S_r| = |T_{r'}|$ but $\#r < \#r'$ where $\#r$ and $\#r'$ is the number of children of r and r'
4) $(G[X_r], \sigma) = (H[X'_{r'}], \sigma')$ and $|S_r| = |T_{r'}|$ and $\#r = \#r' = l$ but $(S_{s_1}, \ldots, S_{s_l}) <_{\mathtt{T}} (T_{t_1}, \ldots, T_{t_l})$ where we assume that $S_{s_1} \leq_{\mathtt{T}} \cdots \leq_{\mathtt{T}} S_{s_l}$ and $T_{t_1} \leq_{\mathtt{T}} \cdots \leq_{\mathtt{T}} T_{t_l}$ are ordered subtrees of S_r and $T_{r'}$, respectively. To compute the order between the subtrees $S_{s_i} \leq_{\mathtt{T}} T_{t_j}$ we consider
   - *i***:** the lexicographical order of the minimal separating sets (s_i and t_j) in X_r and $X'_{r'}$ according to σ and σ', as the primary criterion (observe that the separating sets are subsets of X_r (resp. $X_{r'}$) and are therefore ordered by σ and σ') and
   - *ii***:** pairwise the children $a_{i,i'}$ of s_i and $a'_{j,j'}$ of t_j (for all i' and j' via cross-comparisons) such that the induced bipartite graphs $B_G[s_i, a_{i,i'}]$ and

$B_H[t_j, a'_{j,j'}]$ can be *matched* according to σ and σ' (i.e. $\sigma\sigma'^{-1}$ is an isomorphism) and

iii: *recursively* the subtrees rooted at the children of s_i and t_j. Note, that these children are again bag nodes. For the cross camparison of bag nodes $a_{i,i'}$ and $a'_{j,j'}$ we restrict the set $\Theta_{(a_{i,i'},a'_{j,j'})}$ to a subset of $Sym(X_{a_{i,i'}}) \times Sym(X'_{a'_{j,j'}})$. Namely, $\Theta_{(a_{i,i'},a'_{j,j'})}$ contains the pair $(\phi, \phi') \in Sym(X_{a_{i,i'}}) \times Sym(X'_{a'_{j,j'}})$ if $\phi\phi'^{-1}$ extends the partial isomorphism $\sigma\sigma'^{-1}$ from child $a_{i,i'}$ onto $a'_{j,j'}$ blockwise and which induces an isomorphism from $B_G[s_i, a_{i,i'}]$ onto $B_H[t_j, a'_{j,j'}]$.

We say that two augmented trees S_r and $T_{r'}$ are *equal according to the isomorphism order*, denoted $S_r =_{\mathtt{T}} T_{r'}$, if neither $S_r <_{\mathtt{T}} T_{r'}$ nor $T_{r'} <_{\mathtt{T}} S_r$ holds.

Isomorphism of two subtrees rooted at bag nodes r and r'. We have constant size components associated to the bag nodes. A logspace machine can easily run through all the mappings of X_r and $X'_{r'}$ and record the mappings which gives the minimum isomorphism order. This can be done with cross-comparison of trees (S_r, σ) and $(T_{r'}, \sigma')$ with all possible mappings σ, σ'. Later we will see, that in recursion not all possible mappings for σ and σ' are considered. Observe that $|Sym(X_r)| \in O(1)$.

The comparison of (S_r, σ) and $(T_{r'}, \sigma')$ itself can be done simply by renaming the vertices of X_r and X'_r according to the mappings σ and σ' and then comparing the ordered sequence of edges lexicographically. When equality is found then we recursively compute the isomorphism order of the subtrees rooted at the children of r and r'.

Isomorphism of two subtrees rooted at separating set nodes s_i and t_j. Datta et.al. [5] decompose biconnected planar graphs into triconnected components and obtain a tree on these components and separating pairs, i.e. separating sets of size two. We have separating sets of arbitrary constant size.

Since s_i and t_j correspond to subgraphs of X_r and $X'_{r'}$, we have an order for them given by the fixed mappings σ and σ'. Therefore, we can order the children $s_1, \dots, s_l$ and $t_1, \dots, t_l$ according to their occurrence in X_r and $X'_{r'}$ (e.g. assume $s_i = (1, 2, 3, 7)$ according to the mapping σ and also $s_j = (1, 2, 4, 7)$, then we get $(s_i, \sigma) <_{\mathtt{T}} (s_j, \sigma)$). Hence, when comparing s_i with t_j we have to check whether both come on the same position in that order of $s_1, \dots, s_l$ and $t_1, \dots, t_l$. If so, then we go to the next level in the tree, to the children of s_i and t_j.

Now we have a cross comparison among the children of s_i and the children of t_j. In Steps 4*i*, 4*ii* and 4*iii* we partition the children $a_{i,1}, \dots, a_{i,l_i}$ of s_i and $a'_{j,1}, \dots, a'_{j,l_j}$ of t_j, respectively, into isomorphism classes, step by step.

The membership of a child to a class according to Step 4*i* and 4*ii* can be recomputed. It suffices to keep counters on the work-tape to notice the current class and traversing the siblings from left to right. After these two steps, $a_{i,i'}$ and $a'_{j,j'}$ are in the same class if and only if vertices of s_i and t_j appear lexicographically at the same positions in σ and σ' and the bipartite graphs $B[s_i, a_{i,i'}]$ and $B[t_j, a'_{j,j'}]$ are isomorphic where s_i is mapped onto t_j blockwise corresponding to $\sigma\sigma'^{-1}$ in an isomorphism. In Step 4*iii* we go into recursion and compare members of one class which are rooted at subtrees of the same size. When going into recursion at $a_{i,i'}$ and $a'_{j,j'}$ we consider only those mappings from $(\phi, \phi') \in \Theta_{(a_{i,i'},a'_{j,j'})}$ which induce an isomorphism $\phi\phi'^{-1}$ from $B[s_i, a_{i,i'}]$ onto $B[t_j, a'_{j,j'}]$.

Correctness of the isomorphism order. Both, the bag nodes and the separating set nodes correspond to subgraphs which are basically separating sets. A bag separates all its subtrees from the root and the separating set nodes refine the bag to separating sets of minimum size. Hence, a partial isomorphism is constructed and extended from each node to its child nodes, traversing the augmented tree (the whole graph, accordingly) in depth first manner. In the recursion, the isomorphism between the roots of the current subtrees, say S_r and $T_{r'}$, is partially fixed by the partial isomorphism between their parents. With an exhaustive search we check every possible remaining isomorphism from X_r onto $X'_{r'}$ and go into recursion again partially fixing the isomorphism for the subtrees rooted at children of r and r'. By an inductive argument, the partial isomorphism described for the augmented tree can be followed simultaneously in the original graph and we get:

Theorem 3.3. *The graphs G and H of bounded tree distance width are isomorphic if and only if there is a choice of a root bag r and r' producing augmented trees S_r and $T_{r'}$ such that $S_r =_{\mathtt{T}} T_{r'}$. The isomorphism order between two augmented trees of G and H can be computed in logspace.*

The proof is based on a careful space analysis at each computational step building on concepts of the isomorphism order algorithm of Lindell [12]. The isomorphism order is the basis for a canonization procedure. This is shown in a full version of this paper.

Theorem 3.4. *A graph of bounded tree distance width can be canonized in logspace.*

4. Graphs of bounded treewidth

In this section we consider several isomorphism problems for graphs of bounded treewidth. We are interested in isomorphisms *respecting* the decompositions (i.e. vertices are mapped blockwise from a bag to another bag). We show first that if the tree decomposition of both input graphs is part of the input then the isomorphism problem can be decided in L. We also show that if a tree decomposition of only one of the two given graphs is part of the input, then the isomorphism problem is in LogCFL. It follows that the isomorphism problem for graphs of bounded treewidth is also in LogCFL.

Assume the decompositions of both input graphs are given. Let $(G, D), (H, D')$ be two bounded treewidth graphs together with tree decompositions D and D', respectively. We look for an isomorphism between G and H satisfying the condition that the images of the vertices in one bag in D belong to the same bag in D'.

We prove that this problem is in L. For this we show that given tree decompositions together with designated bags as roots for G and H the question of whether there is an isomorphism between the graphs mapping root to root and respecting the decompositions (i.e. mapping bags in G blockwise onto bags in H) can be reduced to the isomorphism problem for graphs of bounded tree distance decomposition. We argued in the previous section that this problem belongs to L.

Theorem 4.1. *The isomorphism problem for bounded treewidth graphs with given tree decompositions reduces to isomorphism for bounded tree distance width graphs under* AC^0 many-one reductions.

Since bounded tree distance width GI is in L, this almost proves the desired result. To obtain it, we have to find roots for the tree decompositions. We fix an arbitrary bag in the one graph and try all bags from the decomposition of the other graph as roots. We get:

Corollary 4.2. *For every $k \geq 1$ there is a logarithmic space algorithm that, on input a pair of graphs together with a tree decompositions of width k for each of them, decides whether there is an isomorphism between the graphs, respecting the decompositions.*

4.1. A LogCFL algorithm for isomorphism

We consider now the more difficult situation in which only one of the input graphs is given together with a tree decomposition.

Theorem 4.3. *Isomorphism testing for two graphs of bounded treewidth, when a tree decomposition for one of them is given, can be done in* LogCFL.

Proof. We describe an algorithm which runs on a non-deterministic auxiliary pushdown automaton (NAuxPDA). Besides a read-only input tape and a finite control, this machine has access to a stack of polynomial size and a $O(\log n)$ space bounded work-tape. On the input tape we have two graphs G, H of treewidth k and a tree decomposition $D = (\{X_i \mid i \in I\}, T = (I, F), r)$ for G. For $j \in I$ we define G_j to be the subgraph of G induced on the vertex set $\{v \mid v \in X_i, i \in I \text{ and } i = j \text{ or } i \text{ a descendant of } j \text{ in } T\}$. That is, G_j contains the vertices which are separated by the bag X_j from X_r and those in X_j. We define $D_j = (\{X_i, |, i \in I_j\}, T_j = (I_j, F_j), j)$ as the tree decomposition of G_j corresponding to T_j, the subtree of T rooted at j. We also consider a way to order the children of a node in the tree decomposition:

Definition 4.4. Let $1, \ldots, l$ be the children of r in the tree T. We define the *lexicographical subtree order*, as the order among the subtrees $(G_1, D_1), \ldots, (G_l, D_l)$ which is given by: $(G_i, D_i) < (G_j, D_j)$ iff there is a vertex $w \in V(G_i) \setminus X_r$ which has a smaller label than every vertex in $V(G_j) \setminus X_r$.

The algorithm non-deterministically guesses two main structures. First, we guess a tree decomposition of width k for H. This is done in a similar way as in the LogCFL algorithm from Wanke [18] for testing that a graph has bounded treewidth. Second, we guess an isomorphism ϕ from G to H by extending partial mappings from bag to bag.

Very simplified, Wanke's algorithm on input a graph H starts guessing a root bag and it guesses then non-deterministically further bags in the decomposition using the pushdown to test that these bags fulfill the properties of a tree decomposition and that every edge in G is included in some bag. Our algorithm simulates Wanke's algorithm as a subroutine. In the description of the new algorithm we concentrate on the isomorphism testing part and hide the details of how to choose the bags. For simplicity the sentence “guess a bag X_j in H according to Wanke's algorithm” means that we simulate the guessing steps from Wanke, checking at the same time that the constructed structure is in fact a tree decomposition. Note, if the bags were not chosen appropriately, then the algorithm would halt and reject.

We start guessing a root bag $X'_{r'}$ of size $\leq k+1$ for a decomposition of H. With $X'_{r'}$ as root bag we guess the tree decomposition D' of H which corresponds to D and its root r. We also construct a mapping ϕ describing a partial isomorphism from the vertices of G onto the vertices of H. At the beginning, ϕ is the empty mapping and we guess an extension of ϕ from X_r onto $X'_{r'}$. The algorithm starts with $a = r$ (and $a' = r'$). Then we describe isomorphism classes for $1, \ldots, l$, the children of a. First, the children of a can be distinguished because $X_1, \ldots, X_l$ may intersect with X_a differently. Second, we further partition the children within one class according to the number of isomorphic siblings in that class. This can be done in logspace with cross comparisons of pairs among $(G_1, D_1), \dots, (G_l, D_l)$, see Corollary 4.2. It suffices to order the isomorphism classes according to the lexicographical subtree order of the members in the classes. We compare then the children of a with guessed children of a' keeping the following information: For each isomorphism class we check whether there is the same number of isomorphic subtrees of a' in H and whether those intersect with $X'_{a'}$, accordingly. For this we use the lexicographical subtree order to go through the isomorphic siblings from left to right, just keeping a pointer to the current child on the work tape. For two such children, say s_1 of a and t_1 of a', we check then recursively whether (G_1, D_1) is isomorphic to the corresponding subgraph of t_1 in H, by an extension of ϕ.

When we go into recursion, we push on the stack $O(\log n)$ bits for a description of X_a and $X'_{a'}$ as well as a description of the partial mapping ϕ from X_a onto $X'_{a'}$.

In general, we do not keep all the information of ϕ on the stack. We only have the partial isomorphism $\phi : \{v \mid v \in X_r \cup \cdots \cup X_a\} \rightarrow \{v \mid v \in X'_{r'} \cup \cdots \cup X'_{a'}\}$, where $r, \dots, a$ ($r', \dots, a'$, respectively) is a simple path in T from the root to the node at the current level of recursion. After we ran through all children of some node we go one level up in recursion and recompute all the other information which is given implicitly by the subtrees from which we returned. Suppose now, we returned to the bag X_a, we have to do the following:

- Pop from the stack the partial isomorphism ϕ of the bags X_a onto $X'_{a'}$
- Compute the lexicographical next isomorphic sibling. For this we consider the partition into isomorphism classes according to ϕ and the lexicographical subtree order of Definition 4.4. Recall, isomorphism testing of two subtrees of X_a can be done in logspace.
- If there is no such sibling then we compute the lexicographical first child of X_a inside the same isomorphism class. From this child of X_a we compute the sibling which is not in the same isomorphism class and which comes next to the right in the lexicographical subtree order.
- If there is neither a further sibling in the same isomorphism class nor a non-isomorphic sibling of higher lexicographical order then we ran through all children of X_a and we are ready to further return one level up in recursion.

Also for $X'_{a'}$ we guess all children in an isomorphism class from left to right in lexicographical subtree order. If there is no further level to go up in recursion then the stack is empty and we halt in an accepting state. Algorithm 1 summarizes the above considerations.

In Line 1, we guess an extension of ϕ to include a mapping from X_a onto $X'_{a'}$. We know the partial isomorphism of their parent bags since this information can be found on the top of the stack. In Line 3, we have e.g. the partition $E_1 = \{T_1, \dots, T_{l_1}\}$, $E_2 = \{T_{l_1+1}, \dots, T_{l_2}\}$ and so on. It can be obtained in logspace by testing isomorphism of the tree structures $(G_1, D_1), \dots, (G_l, D_l)$. Two subtrees rooted at X_i and X_j are in the same isomorphism class iff there is an automorphism in G which maps X_i onto X_j and fixes their parent X_a setwise. In Lines 6 to 9, we guess $X'_{i'}$ in H which corresponds to X_i, we test recursively whether the corresponding subgraphs G_i and $H_{i'}$ are isomorphic with an extension of ϕ. In Line 7, we check whether $X'_{i'}$ fulfills the properties of a correct tree-decomposition as in Wanke's algorithm (i.e. $X'_{i'}$ must be a separating set which separates its split components from the vertices in $X'_{a'} \setminus X'_{i'}$).

To see that the algorithm correctly computes an isomorphism, we make the following observation. A bag X_a is a separating set which defines the connected subgraphs $G_1, \dots, G_l$.

Algorithm 1 Treewidth Isomorphism with one tree decomposition

```
Input: Graphs $G, H$, tree decomposition $D$ for $G$, bags $X_a$ in $G$ and $X'_{a'}$ in $H$.
Top of Stack: Partial isomorphism $\phi$ mapping the vertices in the parent bag of $X_a$ onto
    the vertices in the parent bag of $X'_{a'}$.
Output: Accept, if $G$ is isomorphic to $H$ by an extension of $\phi$.

Guess an extension of $\phi$ to a partial isomorphism from $X_a$ onto $X'_{a'}$
if $\phi$ cannot be extended to a partial isomorphism which maps $X_a$ onto $X'_{a'}$ then reject
Let $1, \dots, l$ be the children of $a$ in $T$. Partition the subtrees of $T$ rooted at $1, \dots, l$ into
    $p$ isomorphism classes $E_1, \dots, E_p$
for each class $E_j$ from $j = 1$ to $p$
    for each subtree $T_i \in E_j$ (in lexicographical subtree order)
        guess a bag $X'_{i'}$ in $H$ (in increasing lexicographical subtree order). Let $H_{i'}$ be the
            subgraph of $H$ induced by the vertices in $X'_{i'}$ and by those which are separated
            from $X'_{r'}$ in $H \setminus X'_{i'}$
        if $X'_{i'}$ is not a correct child bag of $X'_{a'}$ (see Wanke's algorithm) then reject.
        Invoke this algorithm with input $(G_i, H_{i'}, D_i, X_i, X'_{i'})$ recursively and push $X_a$, $X'_{a'}$
            and the partial isomorphism $\phi$ on the stack
        After recursion pop these informations from the stack
if the stack is not empty then go one level up in recursion
accept and halt
```

These subgraphs do not contain the root X_r and $V(G_i) \cap V(G_j) \subseteq X_a$ since we have a tree decomposition D ($V(G_i)$ are the vertices of G_i). We guess and keep from the partial isomorphism ϕ exactly those parts which correspond to the path from the roots X_r and $X'_{r'}$ to the current bags X_a and $X'_{a'}$. Once we verified a partial isomorphism from one child component (e.g. G_i) of X_a onto a child component (e.g. $H_{i'}$) of $X'_{a'}$, for the other child components it suffices to know the partial mapping of ϕ from X_a onto $X'_{a'}$.

Observe that for each v in G in a computation path from the algorithm there can only be a value for $\phi(v)$. Clearly, if G and H are isomorphic then the algorithm can guess the decomposition of H which fits to D, and the extensions of ϕ correctly. In this case the NAuxPDA has some accepting computation. On the other hand, if the input graphs are non-isomorphic then in every non-deterministic computation either the guessed tree decomposition of H does not fulfill the conditions of a tree decomposition (and would be detected) or the partial isomorphism ϕ cannot be extended at some point. ∎

Wanke's algorithm decides in LogCFL whether the treewidth of a graph is at most k by guessing all possible tree decompositions. Using a result from [7] it follows that there is also a (functional) LogCFL algorithm that on input a bounded treewidth graph computes a particular tree decomposition for it. Since LogCFL is closed under composition, from this result and Theorem 4.3 we get:

Corollary 4.5. *The isomorphism problem for bounded treewidth graphs is in* LogCFL.

Conclusions and open problems. We have shown that the isomorphism problem for graphs of bounded treewidth is in the class LogCFL and that isomorphism testing and canonization of bounded tree distance width graphs is complete for L. By using standard techniques in the area it can be shown that the same upper bounds apply for other problems related to isomorphism on these graph classes. For example the automorphism problem or the functional versions of automorphism and isomorphism can be done within the same complexity classes. The main question remaining is whether the LogCFL upper bound for isomorphism of bounded treewidth graphs can be improved. On the one hand, no LogCFL-hardness result for the isomorphism problem is known, so maybe the result can be improved. We believe that proving a logspace upper bound for the isomorphism problem of bounded treewidth graphs would require to compute tree decompositions within logarithmic space, which is a long standing open question. Another interesting open question is whether bounded treewidth graphs can be canonized in LogCFL.

References

[1] V. Arvind, P. Kurur and T.C. Vijayaraghavan, Bounded color multiplicity graph isomorphism is in the #L hierarchy, in *Proc.20th IEEE CCC* (2005) 13–27.

[2] H.L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees, *J. Algorithms* **11** (1990), 631–643.

[3] H.L. Bodlaender, A partial k-arboreum of graphs with bounded treewidth, *Theoretical Computer Science* **209** (1998), 1–45.

[4] H.L. Bodlaender and A. Koster, Combinatorial optimization of graphs of bounded treewidth, *The Computer Journal* (2007), 631–643.

[5] S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf and F. Wagner, Planar graph isomorphism is in Logspace, In *Proc. 24th IEEE CCC* (2009), 203–214.

[6] S. Datta, P. Nimbhorkar, T. Thierauf and F. Wagner, Isomorphism of $K_{3,3}$-free and K_5-free graphs is in Logspace, To appear in *Proc. 29th FSTTCS* (2009).

[7] G. Gottlob, N. Leone and F. Scarcello, Computing LOGCFL certificates, In *Theoretical Computer Science* **270** (2002), 761–777.

[8] M. Grohe and O. Verbitsky, Testing graph isomorphism in parallel by playing a game, In *Proc. 33rd ICALP* (2006), 3–14.

[9] B. Jenner, J. Köbler, P. McKenzie and J. Torán, Completeness results for Graph Isomorphism, *Journal of Computer and System Sciences* **66** (2003) 549–566.

[10] J. Köbler and S. Kuhnert, The isomorphism problem of k-trees is complete for Logspace, In *Proc. 34th MFCS* (2009), 537–448.

[11] J. Köbler, U. Schöning and J. Torán, *The Graph Isomorphism problem*, Birkhäuser (1993).

[12] S. Lindell, A Logspace algorithm for tree canonization, In *Proc. 24th ACM STOC* (1992), 400–404.

[13] E. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, *Journal of Computer and System Sciences* **25** (1982), 42–65.

[14] E. Luks, Parallel algorithms for permutation groups and graph isomorphism. In *Proc. 27th IEEE FOCS* (1986), 292–302.

[15] G. Miller, Isomorphism testing for graphs of bounded genus, In *Proc.12th ACM STOC*, (1980), 225–235.

[16] O. Reingold, Undirected connectivity in logspace In *Journ. of ACM*, **55** (4) (2008).

[17] I. Sudborrough, Time and tape bounded auxiliary pushdown automata. *Mathematical Foundations of Computer Science* (1977), 493–503.

[18] E. Wanke, Bounded tree-width and LOGCFL *Journal of Algorithms* **16** (1994), 470–491.

[19] K. Yamazaki, H.L. Bodlaender, B. de Fluiter and D.M. Thilikos, Isomorphism for Graphs of Bounded Distance Width, *Algorithmica* **24** (1999), 105–127.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 239-250
www.stacs-conf.org

THE TRAVELING SALESMAN PROBLEM UNDER SQUARED EUCLIDEAN DISTANCES

MARK DE BERG [1] AND FRED VAN NIJNATTEN [1] AND RENÉ SITTERS [2] AND GERHARD J. WOEGINGER [1] AND ALEXANDER WOLFF [3]

[1] Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands.
E-mail address: mdberg@win.tue.nl
E-mail address: f.s.b.v.nijnatten@tue.nl
E-mail address: gwoegi@win.tue.nl

[2] Faculty of Economics and Business Administration, VU Amsterdam, the Netherlands.
E-mail address: rsitters@feweb.vu.nl

[3] Lehrstuhl für Informatik I, Universität Würzburg, Germany.
URL: http://www1.informatik.uni-wuerzburg.de/en/staff/wolff_alexander

Abstract. Let P be a set of points in $\mathbb{R}^d$, and let $\alpha \geqslant 1$ be a real number. We define the distance between two points $p, q \in P$ as $|pq|^\alpha$, where $|pq|$ denotes the standard Euclidean distance between p and q. We denote the traveling salesman problem under this distance function by $\textsc{Tsp}(d, \alpha)$. We design a 5-approximation algorithm for $\textsc{Tsp}(2,2)$ and generalize this result to obtain an approximation factor of $3^{\alpha-1} + \sqrt{6}^{\,\alpha}/3$ for $d = 2$ and all $\alpha \geqslant 2$.

We also study the variant Rev-Tsp of the problem where the traveling salesman is allowed to revisit points. We present a polynomial-time approximation scheme for Rev-$\textsc{Tsp}(2, \alpha)$ with $\alpha \geqslant 2$, and we show that Rev-$\textsc{Tsp}(d, \alpha)$ is apx-hard if $d \geqslant 3$ and $\alpha > 1$. The apx-hardness proof carries over to $\textsc{Tsp}(d, \alpha)$ for the same parameter ranges.

1. Introduction

Motivated by a power-assignment problem in wireless networks (see below for a short discussion of this application) Funke et al. [12] studied the following special case $\textsc{Tsp}(d, \alpha)$ of the Traveling Salesman Problem (Tsp) which is specified by an integer $d \geqslant 2$ and a real number $\alpha > 0$. The cities are n points in d-dimensional space $\mathbb{R}^d$, and the distance between two points p and q is $|pq|^\alpha$, where $|pq|$ denotes the standard Euclidean distance between p and q.

- The objective in problem $\textsc{Tsp}(d, \alpha)$ is to find a shortest tour (under distances $|\cdot|^\alpha$) that visits every city *exactly* once.

1998 ACM Subject Classification: I.1.2 Algorithms, F.2.2 Nonnumerical Algorithms and Problems.
Key words and phrases: Geometric traveling salesman problem, power-assignment in wireless networks, distance-power gradient, NP-hard, APX-hard.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE
27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2458

- In the closely related problem Rev-TSP(d, α), the objective is to find a shortest tour that visits every city *at least* once; thus the salesman is allowed to revisit cities.

Note that TSP$(2, 1)$ is the classical two-dimensional Euclidean TSP and that TSP(d, ∞) is the so-called *bottleneck* TSP in $\mathbb{R}^d$, where the goal is to find a tour whose longest edge has minimum length. We are, however, mainly interested in the case where α is some small constant, and we will not touch the case $\alpha = \infty$.

Similarities and differences to the classical Euclidean TSP. The classical Euclidean TSP is NP-hard even in two dimensions, but it is relatively easy to approximate. In particular, it admits a polynomial-time approximation scheme: Given a parameter $\varepsilon > 0$ and a set of n points in d-dimensional Euclidean space, one can find in $2^{(d/\varepsilon)^{O(d)}} + (d/\varepsilon)^{O(d)} n \log n$ time a tour whose length is at most $1 + \varepsilon$ times the optimal length [23].

A crucial property of the Euclidean TSP is that the underlying Euclidean distances satisfy the triangle inequality. The triangle inequality implies that no reasonable salesman would ever revisit the same city: Instead of returning to a city, it is always cheaper to skip the city and to travel directly to the successor city. All positive approximation results for the Euclidean TSP rely heavily on the triangle inequality. In strong contrast to this, for exponents $\alpha > 1$ the distance function $|\cdot|^\alpha$ does not satisfy the triangle inequality. Thus the combinatorial structure of the problem changes significantly—for example, revisits may suddenly become helpful—and the existing approximation algorithms for Euclidean TSP cannot be applied.

Another nice property of the classical Euclidean problem TSP$(2, 1)$ is that, sloppily speaking, instances with many cities have long optimal tours. Consider for instance a set P of n points in the unit square. Then there exists a tour whose Euclidean length is bounded by $O(\sqrt{n})$ [15]. This bound is essentially tight since there are point sets for which *every* tour has Euclidean length $\Omega(\sqrt{n})$. Interestingly, these results do not carry over to TSP$(2, 2)$ with *squared* Euclidean distances. Problem #124 in the book by Bollobás [8] shows that there always exists a tour for P such that the sum of the squared Euclidean distances is bounded by 4, and that this bound of 4 is best possible. Since, as a rule of thumb, large objective values are easier to approximate than small objective values, this already indicates a substantial difference in the approximability behaviors of TSP(2,1) and TSP(2,2).

Previous work and our results. Funke et al. [12] note that the distance function $|\cdot|^\alpha$ satisfies the so-called *τ-relaxed triangle inequality* with parameter $\tau = 2^{\alpha-1}$ (see Section 2 for a definition). The classical TSP under the τ-relaxed triangle inequality has been extensively studied [2, 3, 6, 7], and all the corresponding machinery from the literature can be applied directly to TSP(d, α). For instance, Andreae [6] derives a $(\tau^2 + \tau)$-approximation for the classical TSP under the τ-relaxed triangle inequality (Δ_τ-TSP, for short). This result translates into a $(4^{\alpha-1} + 2^{\alpha-1})$-approximation for TSP$(\cdot, \alpha)$. For $\tau > 3$, it is better to apply Bender and Chekuri's 4τ-approximation [2] for Δ_τ-TSP, which yields a $2^{\alpha+1}$-approximation for TSP$(\cdot, \alpha)$. Funke et al. derive a $(2 \cdot 3^{\alpha-1})$-approximation algorithm for TSP$(\cdot, \alpha)$, which for the range $2 < \alpha < \log_{3/2} 3 \approx 2.71$ is better than applying the known results [6, 2]. The best result for $\alpha < 2$ is obtained by Böckenhauer et al. [7] whose Christofides-based $(3\tau^2/2)$-approximation for Δ_τ-TSP yields a $(3 \cdot 2^{2\alpha-3})$-approximation for TSP$(\cdot, \alpha)$.

We will demonstrate in Section 2 that essentially *every* variant of the original T^3-algorithm by Andreae and Bandelt [3] already gives a $(2 \cdot 3^{\alpha-1})$-approximation for TSP(d, α). The bottom-line of all this, and the actual starting point of our paper, is that the machinery around the τ-relaxed triangle inequality only yields a bound of roughly $2 \cdot 3^{\alpha-1}$. This raises the following questions: How much can geometry help us in getting even better approximation ratios? Can we beat the 6-approximation for TSP$(2,2)$ of Funke et al.? We answer these questions affirmatively: We develop a new variant of the T^3-algorithm which we call the *geometric* T^3*-algorithm.* An intricate analysis in Section 3 shows that this yields a 5-approximation for TSP$(2,2)$. We then extend our analysis to TSP$(2,\alpha)$ with $\alpha > 2$, and thus obtain a $(3^{\alpha-1} + \sqrt{6}^{\alpha}/3)$-approximation; see Section 4. This new bound is always better than the bound $2 \cdot 3^{\alpha-1}$ of Funke et al. and of our analysis of the T^3-algorithm.

Finally, in Section 5, we turn our attention to the following two questions: (a) How does the approximability of TSP behave when we make α larger than one? (b) Does allowing revisits change the complexity or the approximability of the problem? As we know, classical Euclidean TSP (that is, TSP$(d,1)$) is NP-hard [19] and has a polynomial-time approximation scheme (PTAS) in any fixed number d of dimensions [4]. On the other hand, Rev-TSP(d,α) has—to the best of our knowledge—not been studied before. Concerning question (b), complexity behaves as expected: Rev-TSP(d,α) is NP-hard for any $d \geqslant 2$ and any $\alpha > 0$, and our (straightforward) hardness argument also works for TSP(d,α). In terms of approximability, we show that whereas the two-dimensional problem Rev-TSP$(2,\alpha)$ still has a PTAS for all values $\alpha \geqslant 2$, the problem becomes APX-hard for all $\alpha > 1$ in three dimensions. We were surprised that the APX-hardness proof, too, carried over to TSP$(3,\alpha)$ for all $\alpha > 1$. This inapproximability result stands in strong contrast to the behavior of the classical Euclidean TSP (the case $\alpha = 1$).

The connection to wireless networks. Consider a wireless network whose nodes are equipped with omni-directional antennas. The nodes are modeled as points in the plane, and every node can communicate with all other nodes that are within its transmission radius. The power (that is, the energy) needed to achieve a transmission radius of r is roughly proportional to r^{α} for some real parameter α called the *distance-power gradient.* Depending on environmental conditions, α typically is in the range 2 to 6 [13, Chapter 1]. The goal is to assign powers to the nodes such that the resulting network has certain desirable properties, while the overall power consumption is minimized. A widely studied variant has the objective to make the resulting network strongly connected [1, 11, 16]. Other variants (finding broadcast trees; having small hop diameter; etc) have been studied as well. Funke et al. [12] suggest that it is useful to have a tour through the network, which can be used to pass a *virtual token* around. The resulting power-assignment problem is TSP$(2,\alpha)$.

Another setting related to TSP$(2,\alpha)$ is the following. Instead of omni-directional antennas, some wireless networks use directional antennas. This achieves the same transmission radius under a smaller energy consumption [17, 22]. To model directional antennas, Caragiannis et al. [9] assume that a node can communicate with other nodes in a circular sector of a given angle (where the sector's radius is still determined by the power of the node's signal). For directional antennas one not only has to assign a power level to each node, but also has to decide on the direction in which each node transmits. If the opening angle tends to zero and the points are in general position, a strongly connected network becomes a tour. Hence, our results on TSP$(2,\alpha)$ may shed some light on the difficulty of power assignment for directional antennas with small opening angles.

2. Approximating Tsp$(\cdot, \alpha)$

In this section we lay the basis for our main contribution, a 5-approximation for Tsp(2, 2) in Section 3. We review known algorithms for a related version of Tsp, which can be applied to our setting. As it turns out, these algorithms already yield the same worst-case bounds as the algorithm that Funke et al. [12] gave recently.

We recall some definitions. Let S be a set, let $\mathrm{dist}(\cdot,\cdot) : S \times S \to \mathbb{R}_{\geqslant 0}$ be a distance function on S, and let $\tau \geqslant 1$. We say that $\mathrm{dist}(\cdot,\cdot)$ fulfills the *τ-relaxed triangle inequality* if any three elements $p, q, r \in S$ satisfy $\mathrm{dist}(p, r) \leqslant \tau \cdot (\mathrm{dist}(p, q) + \mathrm{dist}(q, r))$. Recall that we denote by Δ_τ-Tsp the Tsp problem on complete graphs whose weight function (when viewed as a distance function on the vertices) fulfills the τ-relaxed triangle inequality. The following lemma, which has been observed by Funke et al. [12], allows us to apply algorithms for Δ_τ-Tsp to our problem. The proof relies on Hölder's inequality.

Lemma 2.1 ([12]). *Let $\alpha > 0$ be a fixed constant. The distance function $|\cdot|^\alpha : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{\geqslant 0}, (p, q) \mapsto |pq|^\alpha$ fulfills the τ-relaxed triangle inequality for $\tau = 2^{\alpha-1}$.*

Andreae and Bandelt [3] gave an approximation algorithm for Δ_τ-Tsp. Their T^3-algorithm is an adaptation of the well-known double-spanning-tree heuristic for Tsp. This heuristic finds a minimum spanning tree (MST) in the given graph G, doubles all edges, finds an Euler tour in the resulting multigraph, and finally constructs a Hamiltonian cycle from the Euler tour by skipping all nodes that have already been visited. The weight of the MST is a lower bound for the length of a Tsp-tour since removing any edge from a tour yields a spanning tree whose weight is at least the weight of the MST. Note that this statement holds for arbitrary weight functions. If the triangle inequality holds, the heuristic yields a 2-approximation since then skipping over visited nodes never increases the length of the tour, which initially equals twice the weight of the MST. For the weight function $|\cdot|^\alpha$, however, the heuristic can perform arbitrarily badly—consider a sequence of n equally-spaced points on a line.

The T^3-algorithm of Andreae and Bandelt also creates a Hamiltonian tour by shortcutting the MST, but their algorithm never skips more than two consecutive nodes. It is never necessary to skip more than two consecutive nodes because the cube T^3 of a tree T is always Hamiltonian by a result of Sekanina [24]. Recall that the cube of a graph G contains an edge uv if there is a path from u to v in G that uses at most three edges. The proof of Sekanina is constructive; Andreae and Bandelt use it to construct a tour in MST^3.

The recursive procedure of Sekanina [24] to obtain a Hamiltonian cycle in T^3 intuitively works as illustrated in Fig. 1; for the pseudo-code, see Algorithm 1. The algorithm is applied to a tree T and an edge $e = u_1u_2$ of T. Removing the edge e splits the tree into two components T_1 and T_2. In each component T_i $(i = 1, 2)$, the algorithm selects an arbitrary edge $e_i = u_iw_i$ incident to u_i and recursively computes a Hamiltonian cycle of T_i that includes the edge e_i. The algorithm returns a Hamiltonian cycle of T that includes e. The cycle consists of the cycles in T_1 and T_2 without the edges e_1 and e_2, respectively. The two resulting paths are stitched together with the help of e and the new edge w_1w_2.

Note that different choices of the edge e_i in line 5 give rise to different versions of the algorithm. The standard T^3-algorithm takes an arbitrary such edge, while Andreae's refined version [2] makes a specific choice, which gives a better result. (In the next section we will choose e_i based on the local geometry of the MST, which will lead to an improved result for our problem.) Andreae's tour in MST^3 has weight at most $(\tau^2 + \tau)$ times the weight of the MST, which is worst-case optimal [3]. Combining his result with Lemma 2.1 yields that

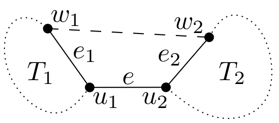


Figure 1: Recursively finding a Hamiltonian cycle in the cube of the tree T.

Algorithm 1: CycleInCube(T, $e = u_1u_2$)

```
1 for $i \leftarrow 1$ to 2 do
2     $T_i \leftarrow$ component of $T - e$ that contains $u_i$
3     if $|T_i| = 1$ then $P_i \leftarrow \emptyset$; $w_i \leftarrow u_i$
4     else
5         pick an edge $e_i = u_iw_i$ incident to $u_i$ in $T_i$
6         if $|T_i| = 2$ then $\Pi_i \leftarrow e_i$
7         else $\Pi_i \leftarrow$ CycleInCube$(T_i, e_i) - e_i$
8 return $\Pi_1 + e + \Pi_2 + w_1w_2$
```

the refined T^3-algorithm is a $(4^{\alpha-1} + 2^{\alpha-1})$-approximation for Tsp$(\,\cdot\,, \alpha)$. We now improve on this with the help of a simple argument. We will frequently use the following definition. Let T be a tree and let $v_0, \dots, v_k$ be a simple path in T. Then we call v_0v_k a *k-shortcut* of T. We say that a shortcut *vw uses* an edge e if e lies on the path connecting v and w in T. It is not hard to see that the weight of a k-shortcut can be bounded as follows.

Lemma 2.2. *Let $\alpha \geqslant 1$ and let e be a k-shortcut using edges $e_1, \dots, e_k$. Then $|e|^\alpha \leqslant k^{\alpha-1} \sum_{i=1}^{k} |e_i|^\alpha$.*

Given a tree T, the tour constructed by the T^3-algorithm consists of edges of T and 2- and 3-shortcuts that use edges of T. Note that in this tour each edge of T is used exactly twice. Thus, for $\alpha \geqslant 2$, the original T^3-algorithm does actually better than the bound we obtained above for the refined T^3-algorithm.

Corollary 2.3. *Every version of the* T^3*-algorithm is a* $(2{\cdot}3^{\alpha-1})$*-approximation for* Tsp$(\,\cdot\,, \alpha)$.

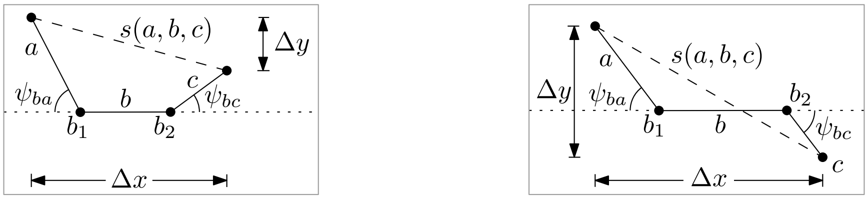
Note that our improved analysis of the T^3-algorithm yields the same result as the algorithm of Funke et al. [12].

Bender and Chekuri [6] designed a 4τ-approximation for Δ_τ-Tsp using a different lower bound: the optimal Tsp tour is a biconnected subgraph of the original graph. The weight of the optimal Tsp tour is at least that of the minimum-weight biconnected subgraph. The latter is np-hard to compute [10], but can be approximated within a factor of 2 [21]. Moreover, the square of a biconnected subgraph is always Hamiltonian. Thus using only edges of the biconnected subgraph and *two*-shortcuts yields a 4τ-approximation for Δ_τ-Tsp. Combining the result of Bender and Chekuri with Lemma 2.1 immediately yields the following result, which is better than Corollary 2.3 for $\alpha > \log_{3/2} 3 \approx 2.71$.

Corollary 2.4. *The algorithm of Bender and Chekuri is a* $2^{\alpha+1}$*-approximation for* Tsp$(\,\cdot\,, \alpha)$.

3. A 5-Approximation for Tsp(2,2)

In the previous section we have used graph-theoretic arguments to determine the performance of the T^3-algorithm. By Corollary 2.3, the T^3-algorithm yields a 6-approximation for $\alpha = 2$, independently of the dimension of the underlying Euclidean space. We now define what we call the *geometric* T^3-algorithm and show that it yields a 5-approximation for Tsp$(2, 2)$. The geometric T^3-algorithm simply chooses in line 5 of Algorithm 1 the edge e_i that makes the smallest angle with the edge e.



(a) a and c lie on the same side of the line through b

(b) a and c lie on different sides of the line through b

Figure 2: Two cases for computing the length of the 3-shortcut $s(a, b, c)$.

The idea behind taking advantage of geometry is as follows. In Corollary 2.3 we have exploited the fact that each edge is used in two ($\leqslant 3$)-shortcuts. The weight of a 3-shortcut is maximum if the corresponding points lie on a line. For the case of the Euclidean MST it is well-known that edges make an angle of at least $\pi/3$ if they share an endpoint. The same proof also works for the MST w.r.t. $|\cdot|^\alpha$. This guarantees that in line 5 of Algorithm 1, we can pick an edge e_i that makes a relatively small angle with e—if the degree of u_i is larger than 2. Otherwise, it is easy to see that e_i is used by a ($\leqslant 2$)- and a ($\leqslant 3$)-shortcut, which is favorable to being used by two 3-shortcuts, see Lemma 2.2.

Although the intuition behind our geometric T^3-algorithm is clear, its analysis turns out to be non-trivial. We start with the following lemma that can be proved with some elementary trigonometry. Given two line segments s and t incident to the same point, we denote the smaller angle between s and t by $\angle st$ and define $\psi_{st} = \pi - \angle st$.

Lemma 3.1. *Given a tree T, the 3-shortcut $s(a, b, c)$ that uses the edges a, b, c of T in this order has weight*

$$|s(a,b,c)|^2 \quad = \quad |a|^2 + |b|^2 + |c|^2 + 2|a||b|\cos\psi_{ba} + 2|b||c|\cos\psi_{bc} + 2|a||c|\cos(\psi_{ba} + \delta\cdot\psi_{bc}),$$

where $\delta = +1$ if a and c lie on the same side of the line through b, and $\delta = -1$ if a and c lie on opposite sides. Moreover, $|s(a,b,c)|^2 \leqslant 2|a|^2 + |b|^2 + 2|c|^2 + 2|a||b|\cos\psi_{ba} + 2|b||c|\cos\psi_{bc}$.

Lemma 3.1 (illustrated in Fig. 2) expresses the weight of a 3-shortcut in terms of the lengths of the edges and the angles between them. Now we show that if an edge a is used in two 3-shortcuts, two of these angles are related. Note that the T^3-algorithm generates the two 3-shortcuts that use a in two consecutive recursive calls, see Fig. 3. The T^3-algorithm is first applied to edge b and then recursively to edge a. In the recursive call, the shortcut $s(e, a, d)$ is generated where d is an edge incident to both a and b. Then the algorithm returns from the recursion and generates the 3-shortcut $s(a, b, c)$. Thus a is the middle edge in one 3-shortcut and the first or last edge in the other 3-shortcut. We rely on the following.

Lemma 3.2. *If the geometric* T^3*-algorithm generates the two 3-shortcuts $s(a, b, c)$ and $s(e, a, d)$ in two recursive calls and d is incident to both a and b, then $\psi_{ba} \geqslant (\pi - \psi_{ad})/2$.*

Now we are ready to prove the main result of this section.

Theorem 3.3. *The geometric* T^3*-algorithm yields a 5-approximation for* TSP$(2, 2)$.

Proof. We express the length of each shortcut s of the T^3-tour in terms of the lengths of the MST edges that s uses. Changing the perspective, for each MST edge a, we use contrib(a) to denote the sum of all terms that contain the factor $|a|$. The edge a is used in at most two shortcuts. Bounding their lengths yields an upper bound on contrib(a). The sum of all contributions relates the length of the T^3-tour to that of the MST (w.r.t. $|\cdot|^\alpha$), which in turn is a lower bound for the length of an optimal TSP tour.

Due to Lemma 2.2, contrib$(a) \leqslant 5|a|^2$ if a is used in a ($\leqslant$2)-shortcut on one side and a ($\leqslant$3)-shortcut on the other side. So we focus on the case that a is used in two 3-shortcuts, see Fig. 3. We rewrite the composite terms in the bound for $s(a,b,c)$ in Lemma 3.1 using Young's inequality with ε, which, given $x, y \in \mathbb{R}$ and $\varepsilon > 0$, states that $xy \leqslant x^2/(2\varepsilon) + y^2\varepsilon/2$.

Let v be the vertex that is incident to edges a and b. If there are multiple 3-shortcuts that use edges that are incident to v then the T^3-algorithm generates these in consecutive recursive calls. We renumber the edges incident to v such that the algorithm is first applied to vv_1, then recursively to vv_2 etc. Then there is some $i \geqslant 1$ such that $b = vv_i$ and $a = vv_{i+1}$ because the algorithm is first applied to b and then recursively to a. We define $\psi_i = \psi_{vv_i,vv_{i+1}} (= \psi_{ba})$. We rewrite the term $2|a||b|\cos\psi_{ba}$ in the bound for $|s(a,b,c)|^2$ in Lemma 3.1 as follows.

$$2|a||b|\cos\psi_{ba} = 2|vv_i||vv_{i+1}|\cos\psi_i \leqslant f(|vv_{i+1}|, |vv_i|, \psi_i), \tag{3.1}$$

where

$$f(|vv_{i+1}|, |vv_i|, \psi_i) = \begin{cases} 0 & \text{if } \psi_i \geqslant \frac{\pi}{2}, \\ |vv_i|^2 + |vv_{i+1}|^2\cos^2\psi_i & \text{if } \psi_i < \frac{\pi}{2} \text{ and} \\ & \big(i = 1 \text{ or } \big(i > 1 \text{ and } \psi_{i-1} \geqslant \frac{\pi}{2}\big)\big), \\ \big(|vv_i|^2 + |vv_{i+1}|^2\big)\cos\psi_i & \text{if } \psi_i < \frac{\pi}{2} \text{ and } i > 1 \text{ and } \psi_{i-1} < \frac{\pi}{2}. \end{cases}$$

The second case of inequality (3.1) follows from Young's inequality with $\varepsilon = 1/\cos\psi_i$ and the third case from Young's inequality with $\varepsilon = 1$. Replacing $2|b||c|\cos\psi_{bc}$ in the bound for $|s(a,b,c)|^2$ in Lemma 3.1 is analogous. Together, the two replacements yield the bound

$$|s(a,b,c)|^2 \leqslant 2|a|^2 + |b|^2 + 2|c|^2 + f(|a|, |b|, \psi_{ba}) + f(|c|, |b|, \psi_{bc}). \tag{3.2}$$

We use (3.2) to bound the weights of all 3-shortcuts. The weight of the final tour is the sum of the weights of all shortcuts. In this sum we can take the two occurrences of an edge $a = vv_{i+1}$ together and analyze the contribution of a to the tour. Note that the result of (3.2) is still at most $3(|a|^2 + |b|^2 + |c|^2)$. So if an edge a is used in a ($\leqslant$3)-shortcut on one side and a ($\leqslant$ 2)-shortcut on the other side, then we still have that contrib$(a) \leqslant 5|a|^2$. It remains to consider the case that a is used in two 3-shortcuts. Let $s(a,b,c)$ and $s(e,a,d)$ be these 3-shortcuts. The algorithm is first applied to edge b and generates shortcut $s(a,b,c)$, where a is the first or the third edge of the shortcut. Then the algorithm is recursively applied to edge a and generates shortcut $s(e,a,d)$, where a is the middle edge. Fig. 3 shows how the vertices are numbered in this case.

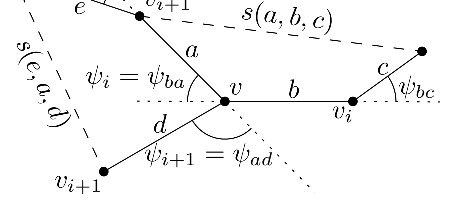


Figure 3: Two 3-shortcuts that use edge a.

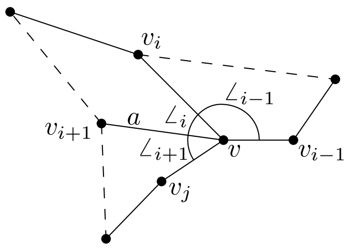


Figure 4: Illustration of case III.

Let σ_a be a function that takes a sum of terms and returns the sum of all terms that contain $|a|$. We derive the following expression for contrib(a).

$$\begin{aligned}\mathrm{contrib}(a) &= \sigma_a(\mathrm{weight}(s(a,b,c))) + \sigma_a(\mathrm{weight}(s(e,a,d)))\\ &\leqslant \sigma_a\left(2|a|^2 + |b|^2 + 2|c|^2 + f(|a|,|b|,\psi_{ba}) + f(|c|,|b|,\psi_{bc})\right)\\ &\quad + \sigma_a\left(2|e|^2 + |a|^2 + 2|d|^2 + f(|e|,|a|,\psi_{ae}) + f(|d|,|a|,\psi_{ad})\right)\\ &\leqslant 4|a|^2 + \sigma_a(f(|vv_{i+1}|,|vv_i|,\psi_i)) + \sigma_a(f(|vv_{i+2}|,|vv_{i+1}|,\psi_{i+1})) \qquad (3.3)\end{aligned}$$

By definition of f we have to consider three cases in (3.3) for contrib(a).

Case I: $\psi_i \geqslant \pi/2$ or $\psi_{i+1} \geqslant \pi/2$.

We assume w.l.o.g. that $\psi_i \geqslant \pi/2$. Then we know that $f(|vv_{i+1}|,|vv_i|,\psi_i)) = 0$ and in the worst case $\sigma_a(f(|vv_{i+2}|,|vv_{i+1}|,\psi_{i+1})) \leqslant |a|^2$. Thus we have that contrib$(a) \leqslant 5|a|^2$.

Case II: $\psi_i < \pi/2$ and $\psi_{i+1} < \pi/2$ and ($i = 1$ or ($i > 1$ and $\psi_{i-1} \geqslant \pi/2$)).

By definition of f we have:

$$\begin{aligned}\sigma_a(f(|vv_{i+1}|,|vv_i|,\psi_i)) &= \sigma_a\left(|vv_i|^2 + |vv_{i+1}|^2\cos^2\psi_i\right) &= |a|^2\cos^2\psi_i\\ \sigma_a(f(|vv_{i+2}|,|vv_{i+1}|,\psi_{i+1})) &= \sigma_a\left((|vv_{i+1}|^2 + |vv_{i+2}|^2)\cos\psi_{i+1}\right) &= |a|^2\cos\psi_{i+1}\end{aligned}$$

Lemma 3.2 states that $\psi_i \geqslant (\pi - \psi_{i+1})/2$. We also know that $\psi_i \leqslant \pi$ by definition. Thus we have

$$\mathrm{contrib}(a) \leqslant \left(4 + \cos^2\frac{\pi - \psi_{i+1}}{2} + \cos\psi_{i+1}\right)|a|^2 \ \leqslant\ 5|a|^2.$$

Case III: $\psi_i < \pi/2$ and $\psi_{i+1} < \pi/2$ and $i > 1$ and $\psi_{i-1} < \pi/2$.

It can be shown that this leads to a contradiction, see Fig. 4 (on page 245).

In cases I and II, the contribution of any edge $|a|$ to the tour is at most $5|a|^2$. The theorem follows by summing up the contributions of all edges. ∎

When using the MST as a lower bound in the analysis, there is not much room for improvement. There are instances of TSP(2,2) where the T^3-algorithm yields a tour whose weight is $4\frac{4}{11}$ times that of the MST; see also [18, Theorem 4.19].

4. Approximating TSP$(2,\alpha)$ with $\alpha \geqslant 2$

In this section we generalize the main result of the previous section to $\alpha \geqslant 2$. Our new bound is always better than the bound $2 \cdot 3^{\alpha-1}$ of Funke et al. [12], see also Corollary 2.3. For $\alpha < 3.41$ our bound is better than the bound $2^{\alpha+1}$ that follows from the algorithm of Bender and Chekuri [6], see Corollary 2.4.

Theorem 4.1. *The geometric T^3-algorithm yields a $(3^{\alpha-1} + \sqrt{6}^{\alpha}/3)$-approximation for* TSP$(2,\alpha)$ *if $\alpha \geqslant 2$.*

Proof. If an edge a is used in a ($\leqslant$2)-shortcut on one side and a ($\leqslant$3)-shortcut on the other side then the total contribution of a to the tour is at most $(2^{\alpha-1} + 3^{\alpha-1})|a|^\alpha$ by Lemma 2.2. So we will focus our analysis again on the case that a is used in two 3-shortcuts. For $\alpha = 2$ we can express the weight of a 3-shortcut by Lemma 3.1 and rewrite the composite terms as in inequality (3.1). For $\alpha > 2$ we apply Hölder's inequality.

$$\begin{aligned}|s(a,b,c)|^\alpha &= \left(|s(a,b,c)|^2\right)^{\alpha/2}\\ &\leqslant \left(2|a|^2+|b|^2+2|c|^2+f(|a|,|b|,\psi_{ba})+f(|c|,|b|,\psi_{bc})\right)^{\alpha/2}\\ &= \left(\beta_a|a|^2+\beta_b|b|^2+\beta_c|c|^2\right)^{\alpha/2} \qquad (4.1)\\ &\leqslant 3^{\alpha/2-1}\left(\beta_a^{\alpha/2}|a|^\alpha+\beta_b^{\alpha/2}|b|^\alpha+\beta_c^{\alpha/2}|c|^\alpha\right) \qquad (4.2)\end{aligned}$$

We introduced the constants of type β to shorten the expression. Note that the last inequality holds only if $\alpha > 2$.

In order to bound the contribution of an edge a that is used in two 3-shortcuts we follow the proof of Theorem 3.3. Since the assumptions of case III in that proof led to a contradiction, it suffices to consider cases I and II.

Case I: $\psi_i \geqslant \pi/2$ or $\psi_{i+1} \geqslant \pi/2$.

$$\begin{aligned}\operatorname{contrib}(a) &\leqslant 3^{\alpha/2-1}\left((2+\cos\psi_i)^{\alpha/2}+(2+\cos\psi_{i+1})^{\alpha/2}\right)|a|^\alpha\\ &\leqslant 3^{\alpha/2-1}\left(2^{\alpha/2}+3^{\alpha/2}\right)|a|^\alpha \;=\; \left(3^{\alpha-1}+\sqrt{6}^{\,\alpha}/3\right)|a|^\alpha\end{aligned}$$

Case II: $\psi_i < \pi/2$ and $\psi_{i+1} < \pi/2$ and ($i = 1$ or ($i > 1$ and $\psi_{i-1} \geqslant \pi/2$)).

$$\operatorname{contrib}(a) \leqslant 3^{\alpha/2-1}\underbrace{\left((2+\cos\psi_{i+1})^{\alpha/2}+\left(2+\sin^2\psi_{i+1}/2\right)^{\alpha/2}\right)}_{g_\alpha(\psi_{i+1})}|a|^\alpha$$

Now we use the fact that the function $h : [0, 2\pi] \to \mathbb{R}, x \mapsto (2 + \cos x)^k + (2 + \sin^2 x/2)^k$ attains its maximum value at $x = 0$. Thus g_α also attains its maximum in the range $[0, \pi/2)$ in $x = 0$. This yields

$$\operatorname{contrib}(a) \;\leqslant\; 3^{\alpha/2-1}\cdot g_\alpha(0)\cdot|a|^\alpha \;\leqslant\; \left(3^{\alpha-1}+\sqrt{6}^{\,\alpha}/3\right)|a|^\alpha.$$

In both cases we showed that $\operatorname{contrib}(a) \leqslant (3^{\alpha-1} + \sqrt{6}^{\,\alpha}/3)|a|^\alpha$. The theorem follows for $\alpha > 2$ by summing up the contributions of all edges. The case $\alpha = 2$ corresponds to Theorem 3.3. ■

5. The Approximability of Tsp and Rev-Tsp

In this section we discuss complexity and approximability of Tsp and its variant Rev-Tsp, where the salesman is allowed to revisit the cities. Recall that for any fixed dimension $d \geqslant 2$, Tsp$(d, 1)$ is NP-hard [19] and admits a PTAS [4].

Theorem 5.1. *Tsp(d, α) and Rev-Tsp(d, α) are NP-hard for any $d \geqslant 2$ and $\alpha > 0$.*

Proof. Itai et al. [14] showed that, given n points in the unit grid, it is NP-hard to decide whether there is a Tsp tour of Euclidean length n. Thus for both of our problems it is NP-hard to distinguish between OPT $= n$ and OPT $\geqslant n - 1 + \sqrt{2}^{\,\alpha}$. ■

Theorem 5.2. *$\text{TSP}(d, \alpha)$ and Rev-$\text{TSP}(d, \alpha)$ are* APX*-hard for any $d \geqslant 3$ and any $\alpha > 1$.*

Proof. We only discuss the case $d = 3$ and $\alpha = 2$—all other cases can be settled by slightly modified arguments—TSP. and we only consider Rev-TSP; a similar reduction can be used for TSP. We reduce from $\{1, 2\}$-TSP, the TSP on the complete graph where the weight of every edge is either 1 or 2; this problem is APX-hard [20]. An instance of $\{1, 2\}$-TSP consists of the complete graph $K_n = (V_n, E_n)$ with vertex set $V_n = \{v_1, \ldots, v_n\}$, edge set $E_n = \{e_1, \ldots, e_m\}$ where $m = n(n-1)/2$, and edge lengths that are specified by a weight function $w : E_n \to \{1, 2\}$. Given K_n and w, we construct a corresponding instance $P_{n,w} \subset \mathbb{R}^3$ of Rev-$\text{TSP}(3, 2)$.

We start our construction by introducing several auxiliary line segments. For each vertex $v_i \in V_n$ we define its *spine* to be the vertical line segment going from point (ni, ni, n) to point (ni, ni, nm). For each edge $e_k = v_i v_j \in E_n$ with $i < j$, we define two corresponding line segments that are parallel to the xy-plane and that are called *bones.* The first bone connects point (ni, ni, nk) on the spine of v_i to the point (nj, ni, nk). The other bone connects point (nj, nj, nk) on the spine of v_j to the point $(nj, ni - \delta_k, nk)$, where $\delta_k = 1$ if $w(e_k) = 1$ and $\delta_k = \sqrt{2}$ if $w(e_k) = 2$. Note that these two bones do not quite touch; they are separated by a gap of length δ_k.

In order to get the instance $P_{n,w}$ of Rev-$\text{TSP}(3, 2)$, we subdivide every single (spine or bone) line segment introduced above by a dense, evenly distributed set of points—we call these points *cities* from now on—so that every unit-length piece receives n^5 cities. The distance between adjacent cities is $1/n^5$, and so the cost for going from one city to an adjacent city is $1/n^{10}$. All these cities together form instance $P_{n,w}$, and this completes our construction. Since we have introduced line segments with a total length of at most $n \cdot n(m-1) + m \cdot 2n(n-1) < 2n^4$, the overall number of cities is at most $2n^9$.

For $1 \leqslant i \leqslant n$ we call the cities on the spine of v_i and on all bones incident to this spine the *city cluster* of v_i. Traversing all cities within such a city cluster is very cheap; even if we visit every city twice, this costs at most $2 \cdot 2n^9/n^{10} = 4/n$ for all cities in all city clusters together. In a traveling salesman tour, the only expensive steps occur when the salesman jumps from one city cluster to another city cluster. By the above definition of δ_k, when jumping from bone to bone across the gap corresponding to edge e_k the incurred cost is exactly $w(e_k)$. Note that jumping from city cluster to city cluster in any other way would be much more expensive and would thus not reduce the total cost of the tour.

Finally, let us show that our reduction is approximation preserving. Fix an ε with $0 < \varepsilon < 1$. Consider an instance K_n and w of $\{1, 2\}$-TSP, and assume without loss of generality that $n > 4/\varepsilon$. Consider an optimal tour π_0 for this instance. If π_0 uses $\ell \geqslant 0$ edges of length 2 and $n - \ell$ edges of length 1, then it has cost $n + \ell$. Given a PTAS for Rev-TSP, we show how to compute in polynomial time a tour of cost at most $(1 + \varepsilon)(n + \ell)$ for K_n and w.

First note that the tour π_0 can be transformed into a tour π_1 through $P_{n,w}$ that makes ℓ jumps of cost 2 and $n - \ell$ jumps of cost 1. That tour π_1 costs at most $n + \ell + 4/n$. Using our hypothetical PTAS for Rev-TSP, we can compute for any $\varepsilon' > 0$ in polynomial time a tour π_2 through $P_{n,w}$ of cost at most $(1 + \varepsilon')c_{\text{opt}}$, where c_{opt} is the cost of an optimal Rev-TSP tour. The existence of π_1 yields $c_{\text{opt}} \leqslant n + \ell + 4/n$. The tour π_2 can be transformed into a tour π_3 through K_n: Just map the jumps of π_2 to the corresponding edges of K_n. Since this mapping cannot increase the cost, tour π_3 costs at most $(1 + \varepsilon')(n + \ell + 4/n)$.

Choosing $\varepsilon' = \varepsilon/2$ and using $4/n < \varepsilon < 1$, we can bound the cost of π_3 from above by

$$\left(1+\frac{\varepsilon}{2}\right)(n+\ell) + \left(1+\frac{\varepsilon}{2}\right)\varepsilon \;=\; \left(1+\frac{\varepsilon}{2}\right)(n+\ell) + \frac{\varepsilon}{2}(2+\varepsilon) \;=\; (1+\varepsilon)(n+\ell)$$

as desired. Like π_2, the tour π_3 may visit vertices more than once. This can be fixed by greedily introducing shortcuts. The shortcuts do not increase the cost of the tour since the weight function w (trivially) fulfills the triangle inequality. ■

Theorem 5.3. *There exists a PTAS for Rev-*TSP$(2,\alpha)$ *for any* $\alpha \geqslant 2$.

Proof. Given a set P of points in the plane, consider the *Gabriel graph* G_P that has a vertex for each point in P. There is an edge between points p and q, if the open disk with diameter pq is empty, in other words, if for all points $r \in P \setminus \{p,q\}$, the angle $\angle prq$ is at most $\pi/2$. The weight of the edge is $|pq|^\alpha$. Note that $|pr|^\alpha + |rq|^\alpha \leqslant |pq|^\alpha$ if $\angle prq$ is at least $\pi/2$. Therefore, there is an optimal TSP tour with revisits through P that only uses the edges of G_P: Indeed, if a tour uses an edge pq for which there is a point r with $\angle prq \geqslant \pi/2$, then replacing pq by pr and rq would shorten the tour. Such a replacement is feasible since revisiting city r is allowed. The Gabriel graph is planar. Hence we end up with an instance of the TSP on weighted planar graphs, for which a PTAS is known [5]. ■

Recall that a *quasi*-PTAS is an approximation scheme with running time $n^{\operatorname{polylog} n}$, where n is the size of the input. The following result follows immediately from the facts that (a) the metric $|\cdot|^\alpha$ has bounded doubling dimension and (b) TSP on metrics of bounded doubling dimension admits a quasi-PTAS [25].

Theorem 5.4. *There exists a quasi-PTAS for Rev-*TSP(d,α) *for any* $\alpha \in (0,1]$ *and* $d \geqslant 1$.

6. Conclusions

In order to construct considerably better approximation algorithms for TSP(d,α), we expect that substantially different methods of analysis have to be found. A result of Van Nijnatten [18, Theorem 4.19] indicates that there is not much room left for improvement as long as we compare to the MST.

The approximability of Rev-TSP$(2,\alpha)$ for $1 < \alpha < 2$ is an interesting open question. We believe that a (quasi)-PTAS may be obtained using the framework of the PTAS for weighted planar graph TSP by Arora et al. [5]. A simple reduction shows that deriving a PTAS for our problem is at least as hard as deriving a PTAS for weighted planar graph TSP. Assume we have a PTAS for Rev-TSP$(2,\alpha)$ for some $\alpha > 1$. Given a weighted planar graph and a planar embedding, we replace each edge by a dense set of points such that traversing a subedge basically costs zero. By making one subedge of each edge e longer, we can make the cost of that subedge (and thus of e) in Rev-TSP proportional to the weight of e. Then, the costs of the optimal solutions of the two problems will be the same up to an arbitrarily small constant factor of $1+\varepsilon$. Such a reduction is polynomially bounded if all weights are polynomially bounded, which can be achieved by a standard rounding scheme.

A PTAS for Rev-TSP$(2,\alpha)$ for any $\alpha > 1$ would be an interesting generalization of the existing PTAS's for weighted planar graphs. Ideally, one would have a PTAS with running time independent of α since it would contain both Euclidean TSP and weighted planar graph TSP as special cases.

References

[1] E. Althaus, G. Călinescu, I. Mandoiu, S. Prasad, N. Tchervenski, and A. Zelikovsky. Power efficient range assignment for symmetric connectivity in static ad hoc wireless networks. *Wireless Networks*, 12(3):287–299, 2006.

[2] T. Andreae. On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality. *Networks*, 38(2):59–67, 2001.

[3] T. Andreae and H.-J. Bandelt. Performance guarantees for approximation algorithms depending on parametrized triangle inequalities. *SIAM J. Discrete Math.*, 8(1):1–16, 1995.

[4] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[5] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proc. 9th Annu. ACM-SIAM Symp. Discr. Algo.*, p. 33–41, 1998.

[6] M. A. Bender and C. Chekuri. Performance guarantees for the TSP with a parameterized triangle inequality. *Inform. Process. Lett.*, 73(1-2):17–21, 2000.

[7] H.-J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, and W. Unger. Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem. *Theor. Comput. Sci.*, 285(1):3–24, 2002.

[8] B. Bollobás. *The Art of Mathematics – Coffee Time in Memphis*. Cambridge Univ. Press, 2006.

[9] I. Caragiannis, C. Kaklamanis, E. Kranakis, D. Krizanc, and A. Wiese. Communication in wireless networks with directional antennas. In F. M. auf der Heide and N. Shavit, editors, *Proc. 20th Annu. ACM Symp. Parallel Algorithms Architect.*, p. 344–351, 2008.

[10] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM J. Comput.*, 5(4):653–665, 1976.

[11] B. Fuchs. On the hardness of range assignment problems. *Networks*, 52(4):183–195, 2008.

[12] S. Funke, S. Laue, R. Naujoks, and Z. Lotker. Power assignment problems in wireless communication: Covering points by disks, reaching few receivers quickly, and energy-efficient travelling salesman tours. In *Proc. 4th Int. IEEE Conf. Distributed Comput. Sensor Systems*, LNCS 5067, p. 282–295, 2008.

[13] L. Godara. *Handbook of Antennas in Wireless Communications*. CRC Press, 2001.

[14] A. Itai, C. Papadimitriou, and J. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 4:676–686, 1982.

[15] R. Karp and J. Steele. Probabilistic analysis of heuristics. In E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys, editors, *The Traveling Salesman Problem*, chapter 6, p. 181–205. John Wiley, 1985.

[16] L. M. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc. Power consumption in packet radio networks. *Theoret. Comput. Sci.*, 243(1–2):289–305, 2000.

[17] A. Nasipuri, K. Li, and U. R. Sappidi. Power consumption and throughput in mobile ad hoc networks using directional antennas. In *Proc. 11th IEEE Conf. Comput. Commun. & Networks*, p. 620–626, 2002.

[18] F. van Nijnatten. Range assignment with directional antennas. Master's thesis, TU Eindhoven, 2008. http://alexandria.tue.nl/extra1/afstversl/wsk-i/nijnatten2008.pdf.

[19] C. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoret. Comput. Sci.*, 4(3):237–244, 1977.

[20] C. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, 1993.

[21] M. Penn and H. Shasha-Krupnik. Improved approximation algorithms for weighted 2- and 3-vertex connectivity augmentation problems,. *J. Algorithms*, 22(1):187–196, 1997.

[22] R. Ramanathan. On the performance of ad hoc networks with beamforming antennas. In *Proc. 2nd ACM Int. Symp. Mobile Ad Hoc Networking & Comput.*, p. 95–105, 2001.

[23] S. B. Rao and W. D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proc. 30th Annu. ACM Symp. Theory Comput.*, pages 540–550, 1998.

[24] M. Sekanina. On an ordering of the set of vertices of a connected graph. *Publications of the Faculty of Science, University of Brno*, 412:137–142, 1960.

[25] K. Talwar. Bypassing the embedding: Algorithms for low dimensional metrics. In *Proc. 36th Annu. ACM Symp. Theory Comput.*, pages 281–290, 2004.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 251-262
www.stacs-conf.org

BEYOND BIDIMENSIONALITY: PARAMETERIZED SUBEXPONENTIAL ALGORITHMS ON DIRECTED GRAPHS

FREDERIC DORN[1] AND FEDOR V. FOMIN[1] AND DANIEL LOKSHTANOV[1] AND VENKATESH RAMAN[2] AND SAKET SAURABH[2]

[1] Department of Informatics, University of Bergen, Bergen, Norway.
E-mail address: `{dorn|fedor.fomin|daniello}@ii.uib.no`

[2] The Institute of Mathematical Sciences, Chennai, India.
E-mail address: `{vraman|saket}@imsc.res.in`

Abstract. In this paper we make the first step beyond bidimensionality by obtaining subexponential time algorithms for problems on directed graphs. We develop two different methods to achieve subexponential time parameterized algorithms for problems on sparse directed graphs. We exemplify our approaches with two well studied problems. For the first problem, k-Leaf Out-Branching, which is to find an oriented spanning tree with at least k leaves, we obtain an algorithm solving the problem in time $2^{\mathcal{O}(\sqrt{k}\log k)}n + n^{\mathcal{O}(1)}$ on directed graphs whose underlying undirected graph excludes some fixed graph H as a minor. For the special case when the input directed graph is planar, the running time can be improved to $2^{\mathcal{O}(\sqrt{k})}n + n^{\mathcal{O}(1)}$. The second example is a generalization of the Directed Hamiltonian Path problem, namely k-Internal Out-Branching, which is to find an oriented spanning tree with at least k internal vertices. We obtain an algorithm solving the problem in time $2^{\mathcal{O}(\sqrt{k}\log k)} + n^{\mathcal{O}(1)}$ on directed graphs whose underlying undirected graph excludes some fixed apex graph H as a minor. Finally, we observe that for any $\varepsilon > 0$, the k-Directed Path problem is solvable in time $\mathcal{O}((1+\varepsilon)^k n^{f(\varepsilon)})$, where f is some function of ε.

Our methods are based on non-trivial combinations of obstruction theorems for undirected graphs, kernelization, problem specific combinatorial structures and a layering technique similar to the one employed by Baker to obtain PTAS for planar graphs.

1. Introduction

Parameterized complexity theory is a framework for a refined analysis of hard (NP-hard) problems. Here, every input instance I of a problem Π is accompanied with an integer parameter k and Π is said to be fixed parameter tractable (FPT) if there is an algorithm running in time $f(k) \cdot n^{\mathcal{O}(1)}$, where $n = |I|$ and f is a computable function. A central problem in parameterized algorithms is to obtain algorithms with running time

1998 ACM Subject Classification: F.2.2; G.2.2.

Key words and phrases: Parameterized Subexponential Algorithms, Directed Graphs, Out-Branching, Internal Out-Branching.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2459

$f(k) \cdot n^{\mathcal{O}(1)}$ such that f is as slow growing function as possible. This has led to the development of various graph algorithms with running time $2^{\mathcal{O}(k)}n^{O(1)}$— notable ones include k-Feedback Vertex Set [7], k-Leaf Spanning Tree [26], k-Odd Cycle Transversal [29], k-Path [4], and k-Vertex Cover [8] in undirected graphs. A natural question was whether we can get *subexponential time* algorithms for these problems, that is, can we have algorithms with running time $2^{o(k)}n^{\mathcal{O}(1)}$. It is now possible to show that these problems do not admit algorithms with running time $2^{o(k)}n^{\mathcal{O}(1)}$ unless the Exponential Time Hypothesis (ETH) [21, 25] fails. Finding algorithms with subexponential running time on general undirected graphs is a trait uncommon to parameterized algorithms.

However, the situation changes completely when we consider problems on topological graph classes like planar graphs or graphs of bounded genus. In 2000, Alber et al. [1] obtained the first parameterized subexponential algorithm on undirected planar graphs by showing that k-Dominating Set is solvable in time $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$. This result triggered an extensive study of parameterized problems on planar and more general classes of sparse graphs like graphs of bounded genus, apex minor-free graphs and H-minor free graphs. All this work led to subexponential time algorithms for several fundamental problems like k-Feedback Vertex Set, k-Edge Dominating Set, k-Leaf Spanning Tree, k-Path, k-r-Dominating Set, k-Vertex Cover to name a few on planar graphs [1, 12, 23], and more generally, on H-minor-free graphs [13, 14, 15]. These algorithms are obtained by showing a combinatorial relation between the parameter and the structure of the input graph and proofs require strong graph theoretic arguments. This graph-theoretic and combinatorial component in the design of subexponential time parameterized algorithms makes it of an independent interest.

Demaine et al. [13] abstracted out the "common theme" among the parameterized subexponential time algorithms on sparse graphs and created the meta-algorithmic theory of Bidimensionality. The bidimensionality theory unifies and improves almost all known previous subexponential algorithms on spare graphs. The theory is based on algorithmic and combinatorial extensions to various parts of Graph Minors Theory of Robertson and Seymour [30] and provides a simple criteria for checking whether a parameterized problem is solvable in subexponential time on sparse graphs. The theory applies to graph problems that are *bidimensional* in the sense that the value of the solution for the problem in question on $k \times k$ grid or "grid like graph" is at least $\Omega(k^2)$ and the value of solution decreases while contracting or sometime deleting the edges. Problems that are bidimensional include k-Feedback Vertex Set, k-Edge Dominating Set, k-Leaf Spanning Tree, k-Path, k-r-Dominating Set, k-Vertex Cover and many others. In most cases we obtain subexponential time algorithms for a problem using bidimensionality theory in following steps. Given an instance (G, k) to a bidimensional problem Π, in polynomial time we either decide that it is an yes instance to Π or the treewidth of G is $\mathcal{O}(\sqrt{k})$. In the second case, using known constant factor approximation algorithm for the treewidth, we find a tree decomposition of width $\mathcal{O}(\sqrt{k})$ for G and then solve the problem by doing dynamic programming over the obtained tree decomposition. This approach combined with Catalan structure based dynamic programming over graphs of bounded treewidth has led to $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$ time algorithm for k-Feedback Vertex Set, k-Edge Dominating Set, k-Leaf Spanning Tree, k-Path, k-r-Dominating Set, k-Vertex Cover and many others on planar graphs [12, 13, 19] and in some cases like k-Dominating Set and k-Path on H-minor free graphs [13, 17]. We refer to surveys by Demaine and Hajiaghayi [14] and

Dorn et al. [18] for further details on bidimensionality and subexponential parameterized algorithms.

While bidimensionality theory is a powerful algorithmic framework on undirected graphs, it remains unclear how to apply it to problems on directed graphs (or digraphs). The main reason is that Graph Minor Theory for digraphs is still in a nascent stage and there are no suitable obstruction theorems so far. For an example, even the first step of the framework does not work easily on digraphs, as there is no unique notion of directed $k \times k$ grid. Given a $k \times k$ undirected grid we can make $2^{\mathcal{O}(k^2)}$ distinct directed grids by choosing orientations for the edges. Hence, unless we can guarantee a lower bound of $\Omega(k^2)$ on the size of solution of a problem for *any* directed $k \times k$ grid, the bidimensionality theory does not look applicable for problems on digraphs. Even the analogue of treewidth for digraphs is not unique and several alternative definitions have been proposed. Only recently the first non-trivial subexponential parameterized algorithms on digraphs was obtained. Alon et al. [3] introduced the method of chromatic coding, a variant of color coding [4], and combined it with divide and conquer to obtain $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$ for k-Feedback Arc Set in tournaments.

Our contribution. In this paper we make the first step beyond bidimensionality by obtaining subexponential time algorithms for problems on sparse digraphs. We develop two different methods to achieve subexponential time parameterized algorithms for digraph problems when the input graph can be embedded on some surface or the underlying undirected graph excludes some fixed graph H as a minor.

Quasi-bidimensionality. Our first technique can be thought of as “bidimensionality in disguise”. We observe that given a digraph D, whose underlying undirected graph $UG(D)$ excludes some fixed graph H as a minor, if we can remove $o(k^2)$ vertices from the given digraph to obtain a digraph whose underlying undirected graph has a constant treewidth, then the treewidth of $UG(D)$ is $o(k)$. So given an instance (D, k) to a problem Π, in polynomial time we either decide that it is an yes instance to Π or the treewidth of $UG(D)$ is $o(k)$. In the second case, as in the framework based on bidimensionality, we solve the problem by doing dynamic programming over the tree decomposition of $UG(D)$. The dynamic programming part of the framework is problem-specific and runs in time $2^{o(k)} + n^{\mathcal{O}(1)}$. We exemplify this technique on a well studied problem of k-Leaf Out-Branching.

We say that a subdigraph T on vertex set $V(T)$ of a digraph D on vertex set $V(D)$ is an *out-tree* if T is an oriented tree with only one vertex r of in-degree zero (called the *root*). The vertices of T of out-degree zero are called *leaves* and every other vertex is called an *internal vertex*. If T is a spanning out-tree, that is, $V(T) = V(D)$, then T is called an *out-branching* of D. Now we are in position to define the problem formally.

> k-Leaf Out-Branching (k-LOB): Given a digraph D with the vertex set $V(D)$ and the arc set $A(D)$ and a positive integer k, check whether there exists an out-branching with at least k leaves.

The study of k-Leaf Out-Branching has been at forefront of research in parameterized algorithms in the last few years. Alon et al. [2] showed that the problem is fixed parameter tractable by giving an algorithm that decides in time $\mathcal{O}(f(k)n)$ whether a strongly connected digraph has an out-branching with at least k leaves. Bonsma and Dorn [6] extended this result to all digraphs, and improved the running time of the algorithm. Recently, Kneis et al. [26] provided a parameterized algorithm solving the problem in time $4^k n^{\mathcal{O}(1)}$. This result was further improved to $3.72^k n^{\mathcal{O}(1)}$ by Daligaut et al. [10]. Fernau et al. [20] showed that for the rooted version of the problem, where apart from the input instance we are also given a root r and one asks for a k-leaf out-branching rooted at r, admits a $\mathcal{O}(k^3)$ kernel. Furthermore they also show that k-LOB does not admit polynomial kernel unless polynomial hierarchy collapses to third level. Finally, Daligault and Thomassé [11] obtained a $\mathcal{O}(k^2)$ kernel for the rooted version of the k-LOB problem and gave a constant factor approximation algorithm for k-LOB.

Using our new technique in combination with kernelization result of [20], we get an algorithm for k-LOB that runs in time $2^{\mathcal{O}(\sqrt{k}\log k)}n + n^{\mathcal{O}(1)}$ for digraphs whose underlying undirected graph is H-minor-free. For planar digraphs our algorithm runs in $2^{\mathcal{O}(\sqrt{k})}n+n^{\mathcal{O}(1)}$ time.

Kernelization and Divide & Conquer. Our second technique is a combination of divide and conquer, kernelization and dynamic programming over graphs of bounded treewidth. Here, using a combination of kernelization and a Baker style layering technique for obtaining polynomial time approximation schemes [5], we reduce the instance of a given problem to $2^{o(k)}n^{\mathcal{O}(1)}$ many new instances of the same problem. These new instances have the following properties: (a) the treewidth of the underlying undirected graph of these instances is bounded by $o(k)$; and (b) the original input is an yes instance if and only if at least one of the newly generated instance is. We exhibit this technique on the k-Internal Out-Branching problem, a parameterized version of a generalization of Directed Hamiltonian Path.

> k-Internal Out-Branching (k-IOB): Given a digraph D with the vertex set $V(D)$ and the arc set $A(D)$ and a positive integer k, check whether there exists an out-branching with at least k internal vertices.

Prieto and Sloper [28] studied the *undirected* version of this problem and gave an algorithm with running time $2^{4k\log k}n^{\mathcal{O}(1)}$ and obtained a kernel of size $\mathcal{O}(k^2)$. Recently, Fomin et al. [22] obtained a vertex kernel of size $3k$ and gave an algorithm for the undirected version of k-IOB running in time $8^k n^{\mathcal{O}(1)}$. Gutin et al. [24] obtained an algorithm of running time $2^{\mathcal{O}(k\log k)}n^{\mathcal{O}(1)}$ for k-IOB and gave a kernel of size of $\mathcal{O}(k^2)$ using the well known method of crown-decomposition. Cohen et al. [9] improved the algorithm for k-IOB and gave an algorithm with running time $49.4^k n^{\mathcal{O}(1)}$. Here, we obtain a subexponential time algorithm for k-IOB with running time $2^{\mathcal{O}(\sqrt{k}\log k)} + n^{\mathcal{O}(1)}$ on directed planar graphs and digraphs whose underlying undirected graphs are apex minor-free.

Finally, we also observe that for any $\varepsilon > 0$, there is an algorithm finding in time $\mathcal{O}((1+\varepsilon)^k n^{f(\varepsilon)})$ a directed path of length at least k (the k-Directed Path problem) in a digraph which underlying undirected graph excludes a fixed apex graph as a minor. The existence of subexponential parameterized algorithm for this problem remains open.

2. Preliminaries

Let D be a digraph. By $V(D)$ and $A(D)$ we represent the vertex set and arc set of D, respectively. Given a subset $V' \subseteq V(D)$ of a digraph D, let $D[V']$ denote the digraph induced by V'. The *underlying graph* $UG(D)$ of D is obtained from D by omitting all orientations of arcs and by deleting one edge from each resulting pair of parallel edges. A vertex u of D is an *in-neighbor* (*out-neighbor*) of a vertex v if $uv \in A(D)$ ($vu \in A(D)$, respectively). The *in-degree* $d^-(v)$ (*out-degree* $d^+(v)$) of a vertex v is the number of its in-neighbors (out-neighbors). We say that a subdigraph T of a digraph D is an *out-tree* if

T is an oriented tree with only one vertex r of in-degree zero (called the *root*). The vertices of T of out-degree zero are called *leaves* and every other vertex is called an *internal vertex*. If T is a spanning out-tree, that is, $V(T) = V(D)$, then T is called an *out-branching* of D. An out-branching (respectively. out-tree) rooted at r is called *r-out-branching* (respectively. r-out-tree). We define the operation of a *contraction of a directed arc* as follows. An arc uv is contracted as follows: add a new vertex u', and for each arc wv or wu add the arc wu' and for an arc vw or uw add the arc $u'w$, remove all arcs incident to u and v and the vertices u and v. We call a loopless digraph D *rooted*, if there exists a pre-specified vertex r of in-degree 0 as a root r and $d^+(r) \geq 2$. The rooted digraph D is called *connected* if every vertex in $V(D)$ is reachable from r by a directed path.

Let G be an undirected graph with the vertex set $V(G)$ and the edge set $E(G)$. For a subset $V' \subseteq V(G)$, by $G[V']$ we mean the subgraph of G induced by V'. By $N(u)$ we denote (open) neighborhood of u that is the set of all vertices adjacent to u and by $N[u] = N(u) \cup \{u\}$. Similarly, for a subset $D \subseteq V$, we define $N[D] = \cup_{v \in D} N[v]$. The *diameter* of a graph G, denoted by *$diam(G)$*, is defined to be the maximum length of a shortest path between any pair of vertices of $V(G)$.

Given an edge $e = uv$ of a graph G, the graph G/e is obtained by contracting the edge uv; that is, we get G/e by identifying the vertices u and v and removing all the loops and duplicate edges. A *minor* of a graph G is a graph H that can be obtained from a subgraph of G by contracting edges. A graph class $\mathcal{C}$ is *minor closed* if any minor of any graph in $\mathcal{C}$ is also an element of $\mathcal{C}$. A minor closed graph class $\mathcal{C}$ is *H-minor-free* or simply *H-free* if $H \notin \mathcal{C}$. A graph H is called an apex graph if the removal of one vertex makes it a planar graph.

A *tree decomposition* of a (undirected) graph G is a pair (X, T) where T is a tree whose vertices we will call *nodes* and $X = (\{X_i \mid i \in V(T)\})$ is a collection of subsets of $V(G)$ such that (a) $\bigcup_{i \in V(T)} X_i = V(G)$, (b) for each edge $vw \in E(G)$, there is an $i \in V(T)$ such that $v, w \in X_i$, and (c) for each $v \in V(G)$ the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of T. The *width* of a tree decomposition $(\{X_i \mid i \in V(T)\}, T)$ equals $\max_{i \in V(T)}\{|X_i| - 1\}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. We use notation $\mathbf{tw}(G)$ to denote the treewidth of a graph G.

A parameterized problem is said to admit a *polynomial kernel* if there is a polynomial time algorithm (where the degree of the polynomial is independent of k), called a *kernelization* algorithm, that reduces the input instance down to an instance with size bounded by a polynomial $p(k)$ in k, while preserving the answer. This reduced instance is called a *$p(k)$ kernel* for the problem. See [27] for an introduction to kernelization.

3. Method I – Quasi Bidimensionality

In this section we present our first approach. In general, a subexponential time algorithm using bidimensionality is obtained by showing that the solution for a problem in question is at least $\Omega(k^2)$ on $k \times k$ (contraction) grid minor. Using this we reduce the problem to a question on graph with treewidth $o(k)$. We start with a lemma which enables us to use the framework of bidimensionality for digraph problems, though not as directly as for undirected graph problems.

Lemma 3.1. *Let D be a digraph such that $UG(D)$ excludes a fixed graph H as a minor. For any constant $c \geq 1$, if there exists a subset $S \subseteq V(D)$ with $|S| = s$ such that $\mathbf{tw}(UG(D[V(D) \setminus S])) \leq c$, then $\mathbf{tw}(UG(D)) = \mathcal{O}(\sqrt{s})$.*

Proof. By [14], for any H-minor-free graph G with treewidth more than r, there is a constant $\delta > 1$ only dependent on H such that G has a $\frac{r}{\delta} \times \frac{r}{\delta}$ grid minor. Suppose $\mathbf{tw}(UG(D)) > \delta(c+1)\sqrt{s}$ then $UG(D)$ contains a $(c+1)\sqrt{s} \times (c+1)\sqrt{s}$ grid as a minor. Notice that this grid minor can not be destroyed by any vertex set S of size at most s. That is, if we delete any vertex set S with $|S| = s$ from this grid, it will still contain a $(c+1) \times (c+1)$ subgrid. Thus, $UG(D[V(D) \setminus S])$ contains a $(c+1) \times (c+1)$ grid minor and hence by [21, Exercise 11.6] we have that $\mathbf{tw}(UG(D[V(D) \setminus S])) > c$. This shows that we need to delete more than s vertices from $UG(D)$ to obtain a graph with treewidth at most c, a contradiction. ∎

Using Lemma 3.1, we show that k-Leaf-Out-Branching problem has a subexponential time algorithm on digraphs D such that $UG(D)$ exclude a fixed graph H as a minor. For our purpose a rooted version of k-LOB will also be useful which we define now. In the Rooted k-Leaf-Out-Branching (R-k-LOB) problem apart from D and k the root r of the tree searched for is also a part of the input and the objective is to check whether there exists an r-out-branching with at least k leaves. We now state our main combinatorial lemma and postpone its proof for a while.

Lemma 3.2. *Let D be a digraph such that $UG(D)$ excludes a fixed graph H as a minor, k be a positive integer and $r \in V(D)$ be the root. Then in polynomial time either we can construct an r-out-branching with at least k leaves in D or find a digraph D' such that following holds.*

- *$UG(D')$ excludes the fixed graph H as a minor;*
- *D has an r-out-branching with at least k leaves if and only if D' has an r-out-branching with at least k leaves;*
- *there exists a subset $S \subseteq V(D')$ such that $|S| = \mathcal{O}(k)$ and $\mathbf{tw}(U(D'[V(D') \setminus S]) \leq c$, c a constant.*

Combining Lemmata 3.1 and 3.2 we obtain the following result.

Lemma 3.3. *Let D be a digraph such that $UG(D)$ excludes a fixed graph H as a minor, k be a positive integer and $r \in V(D)$ be a root. Then in polynomial time either we can construct an r-out-branching with at least k leaves in D or find a digraph D' such that D has an r-out-branching with at least k leaves if and only if D' has an r-out-branching with at least k leaves. Furthermore $\mathbf{tw}(UG(D')) = \mathcal{O}(\sqrt{k})$.*

When a tree decomposition of $UG(D)$ is given, dynamic programming methods can be used to decide whether D has an out-branching with at least k leaves, see [24]. The time complexity of such a procedure is $2^{\mathcal{O}(w \log w)}n$, where $n = |V(D)|$ and w is the width of the tree decomposition. Now we are ready to prove the main theorem of this section assuming the combinatorial Lemma 3.2.

Theorem 3.4. *The k-LOB problem can be solved in time $2^{\mathcal{O}(\sqrt{k} \log k)}n + n^{\mathcal{O}(1)}$ on digraphs with n vertices such that the underlying undirected graph excludes a fixed graph H as a minor.*

Proof. Let D be a digraph where $UG(D)$ excludes a fixed graph H as a minor. We guess a vertex $r \in V(D)$ as a root. This only adds a factor of n to our algorithm. By Lemma 3.3, we can either compute, in polynomial time, an r-out-branching with at least k leaves in D or find a digraph D' with $UG(D')$ excluding a fixed graph H as a minor and $\mathbf{tw}(UG(D')) = \mathcal{O}(\sqrt{k})$. In the later case, using the constant factor approximation algorithm of Demaine et al. [16] for computing the treewidth of a H-minor free graph, we find a tree decomposition of width $\mathcal{O}(\sqrt{k})$ for $UG(D')$ in time $n^{\mathcal{O}(1)}$. With the previous observation that we can find an r-out-branching with at least k leaves, if exists one, in time $2^{\mathcal{O}(\sqrt{k}\log k)}n$ using dynamic programming over graphs of bounded treewidth, we have that we can solve R-k-LOB in time $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$. Hence, we need $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$ to solve the k-LOB problem.

To obtain the claimed running time bound we use the known kernelization algorithm after we have guessed the root r. Fernau et al. [20] gave an $\mathcal{O}(k^3)$ kernel for R-k-LOB which preserves the graph class. That is, given an instance (D, k) of R-k-LOB, in polynomial time they output an equivalent instance (D'', k) of R-k-LOB such that (a) if $UG(D)$ is H-minor free then so is $UG(D'')$; and (b) $|V(D'')| = \mathcal{O}(k^3)$. We will use this kernel for our algorithm rather than the $\mathcal{O}(k^2)$ kernel for R-k-LOB obtained by Daligault and Thomassé [11], as they do not preserve the graph class. So after we have guessed the root r, we obtain an equivalent instance (D'', k) for R-k-LOB using the kernelization procedure described in [20]. Then using the algorithm described in the previous paragraph we can solve R-k-LOB in time $2^{\mathcal{O}(\sqrt{k}\log k)} + n^{\mathcal{O}(1)}$. Hence, we need $2^{\mathcal{O}(\sqrt{k}\log k)}n + n^{\mathcal{O}(1)}$ to solve k-LOB. ∎

Given a tree decomposition of width w of $UG(D)$ for a planar digraph D, we can solve k-LOB using dynamic programming methods in time $2^{\mathcal{O}(w)}n$. This brings us to the following theorem.

Theorem 3.5. [$\star$][1] *The k-LOB problem can be solved in time $2^{\mathcal{O}(\sqrt{k})}n + n^{\mathcal{O}(1)}$ on digraphs with n vertices when the underlying undirected graph is planar.*

3.1. Proof of Lemma 3.2

To prove the combinatorial lemma we need a few recent results from the literature on out-branching problems. We start with some definitions given in [11]. A *cut* of D is a subset S such that there exists a vertex $z \in V(D) \setminus S$ such that z is not reachable from r in $D[V(D) \setminus S]$. We say that D is 2-*connected* if there exists no cut of size one in D or equivalently there are at least two vertex disjoint paths from r to every vertex in D.

Lemma 3.6 ([11]). *Let D be a rooted 2-connected digraph with r being its root. Let α be the number of vertices in D with in-degree at least 3. Then D has an out-branching rooted at r with at least $\alpha/6$ leaves and such an out-branching can be found in polynomial time.*

A vertex $v \in V(D)$ is called a *nice vertex* if v has an in-neighbor which is not its out-neighbor. The following lemma is proved in [11].

Lemma 3.7 ([11]). *Let D be a rooted 2-connected digraph rooted at a vertex r. Let β be the number of nice vertices in D. Then D has an out-branching rooted at r with at least $\beta/24$ leaves and such an out-branching can be found in polynomial time.*

Proof of Lemma 3.2. To prove the combinatorial lemma, we consider two cases based on whether or not D is 2-connected.

Case 1) *D is a rooted 2-connected digraph.*

[1] The proofs marked with [$\star$] will appear in the final version of the paper.

We prove this case in the following claim.

Claim 1. [$\star$] *Let D be a rooted 2-connected digraph with root r and a positive integer k. Then in polynomial time, we can find an out-branching rooted at r with at least k leaves or find a set S of at most $30k$ vertices whose removal results in a digraph whose underlying undirected graph has treewidth one.*

Case 2) *D is not 2-connected.*

Since D is not 2-connected, it has cut vertices, those vertices that separate r from some other vertices. We deal with the cut vertices in three cases. Let x be a cut vertex of D. The three cases we consider are following.

Case 2a) *There exists an arc xy that disconnects at least two vertices from r.*

In this case, we *contract the arc xy*. After repeatedly applying Case 2a), we obtain a digraph D' such that any arc out of a cut vertex x of D' disconnects at most 1 vertex. The resulting digraph D' is the one mentioned in the Lemma. Since we have only contracted some arcs iteratively to obtain D', it is clear that $UG(D')$ also excludes H as a minor. The proof that such contraction does not decrease the number of leaves follows from a reduction rule given in [20]. We provide a proof for completion.

Claim 2. [$\star$] *Let D be a rooted connected digraph with root r, let xy be an arc that disconnects at least two vertices from r and D' be the digraph obtained after contracting the arc xy. Then D has an r-out-branching with at least k leaves if and only if D' has an r-out-branching with at least k leaves.*

Now we handle the remaining cut-vertices of D' as follows. Let $\mathcal{S}$ be the set of cut vertices in D'. For every vertex $x \in \mathcal{S}$, we associate *a cut-neighborhood* $C(x)$, which is the set of out-neighbors of x such that there is no path from r to any vertex in $C(x)$ in $D'[V(D') \setminus \{x\}]$. By $C[x]$ we denote $C(x) \cup \{x\}$. The following observation is used to handle other cases.

Claim 3. *Let $\mathcal{S}$ be the set of cut vertices in D'. Then for every pair of vertices $x, y \in \mathcal{S}$ and $x \neq y$, we have that $C[x] \cap C[y] = \emptyset$.*

Proof. To the contrary let us assume that $C[x] \cap C[y] \neq \emptyset$. We note that $C[x] \cap C[y]$ can only have a vertex $v \in \{x, y\}$. To prove this, assume to the contrary that we have a vertex $v \in C[x] \cap C[y]$ and $v \notin \{x, y\}$. But then it contradicts the fact that $v \in C[x]$, as x doesn't separate v from r due to the path between r and v through y. Thus, either $x \in C(y)$ or $y \in C(x)$. Without loss of generality let $y \in C(x)$. This implies that we have an arc xy and there exists a vertex $z \in C(y)$ such that $z \notin C(x)$. But then the arc xy disconnects at least two vertices y and z from r and hence Case 2a would have applied. This proves the claim. ∎

Now we distinguish cases based on cut vertices having cut-neighborhood of size at least 2 or 1. Let $\mathcal{S}_{\geq 2}$ and $\mathcal{S}_{=1}$ be the subset of cut-vertices of D' having at least two cut-neighbors and exactly one neighbor respectively.

Case 2b) $\mathcal{S}_{\geq 2} \neq \emptyset$.

We first bound $|\mathcal{S}_{\geq 2}|$. Let $A_c = \{xy \mid x \in \mathcal{S}_{\geq 2}, y \in C(x)\}$ be the set of out-arcs emanating from the cut vertices in $\mathcal{S}_{\geq 2}$ to its cut neighbors. We now prove the following structural claim which is useful for bounding the size of $\mathcal{S}_{\geq 2}$.

Claim 4. [$\star$] *If D' has an r-out-branching T' with at least k leaves then D' has an r-out-branching T with at least k leaves and containing all the arcs of A_c, that is, $A_c \subseteq A(T)$. Furthermore such an out-branching can be found in polynomial time.*

We know that in any out-tree, the number of internal vertices of out-degree at least 2 is bounded by the number of leaves. Hence if $|\mathcal{S}_{\geq 2}| \geq k$ then we obtain an r-out-branching T of D' with at least k leaves using Claim 4 and we are done. So from now onwards we assume that $|\mathcal{S}_{\geq 2}| = \ell \leq k-1$.

We now do a transformation to the given digraph D'. For every vertex $x \in \mathcal{S}_{\geq 2}$, we introduce an *imaginary* vertex x^i and add an arc ux^i if there is an arc $ux \in A(D')$ and add an arc x^iv if there is an arc $xv \in A(D')$. Basically we duplicate the vertices in $\mathcal{S}_{\geq 2}$. Let the transformed graph be called D^{dup}. We have the following two properties about D^{dup}. First, no vertex in $\mathcal{S}_{\geq 2} \cup \{x^i | x \in \mathcal{S}_{\geq 2}\}$ is a cut vertex in D^{dup}. We sum up the second property in the following claim.

Claim 5. [$\star$] *The digraph D' has an r-out-branching T with at least k leaves if and only if D^{dup} has an r-out-branching T' with at least $k+\ell$ leaves.*

Now we move on to the last case.

Case 2c) $\mathcal{S}_{=1} \neq \emptyset$.

Consider the arc set $A_p = \{xy \mid x \in \mathcal{S}_{=1}, y \in C(x)\}$. Observe that $A_p \subseteq A(D') \subseteq A(D^{dup})$ and A_p forms a *matching* in D^{dup} because of Claim 3. Let D_c^{dup} be the digraph obtained from D^{dup} by contracting the arcs of A_p. That is, for every arc $uv \in A_p$, the contracted graph is obtained by identifying the vertices u and v as uv and removing all the loops and duplicate arcs.

Claim 6. *Let D_c^{dup} be the digraph obtained by contracting the arcs of A_p in D^{dup}. Then the following holds.*

(1) *The digraph D_c^{dup} is 2-connected;*
(2) *If D_c^{dup} has an r-out-branching T with at least $k+\ell$ leaves then D^{dup} has an r-out-branching with at least $k+\ell$ leaves.*

Proof. The digraph D_c^{dup} is 2-connected by the construction as we have iteratively removed all cut-vertices. If D_c^{dup} has an r-out-branching T with at least $k+\ell$ leaves then we can obtain a r-out-branching with at least $k+\ell$ leaves for D^{dup} by expanding each of the contracted vertices to arcs in A_p. ■

We are now ready to combine the above claims to complete the proof of the lemma. We first apply Claim 1 on D_c^{dup} with $k+\ell$. Either we get an r-out-branching T' with at least $k+\ell$ leaves or a set S' of size at most $30(k+\ell)$ such that $\mathbf{tw}(UG(D_c^{dup}[V(D_c^{dup}) \setminus S]))$ is one. In the first case, by Claims 5 and 6 we get an r-out-branching T with at least k leaves in D'. In the second case we know that there is a vertex set S' of size at most $30(k+\ell)$ such that $\mathbf{tw}(UG(D_c^{dup}[V(D_c^{dup}) \setminus S']))$ is one. Let $S^* = \{u \mid uv \in S', vu \in S', u \in S'\}$ be the set of vertices obtained from S' by expanding the contracted vertices in S'. Clearly the size of $|S^*| \leq 2|S'| \leq 60(k+\ell) \leq 120k = \mathcal{O}(k)$. We now show that the treewidth of the underlying undirected graph of $D^{dup}[V(D^{dup}) \setminus S^*]$ is at most 3. This follows from the observation that $\mathbf{tw}(UG(D_c^{dup}[V(D_c^{dup}) \setminus S']))$ is one. Hence given a tree-decomposition of width one for $UG(D_c^{dup}[V(D_c^{dup}) \setminus S'])$ we can obtain a tree-decomposition for $UG(D^{dup}[V(D^{dup}) \setminus S^*])$ by expanding the contracted vertices. This can only double the bag size and hence the treewidth of $UG(D^{dup}[V(D^{dup}) \setminus S^*])$ is at most 3, as the bag size can at most be 4. Now we take $S = S^* \cap V(D')$ and since $V(D') \subseteq V(D^{dup})$, we have that $tw(UG(D[V(D) \setminus S])) \leq 3$. This concludes the proof of the lemma. ■

4. Method II - Kernelization and Divide & Conquer

In this section we exhibit our second method of designing subexponential time algorithms for digraph problems through the k-Internal Out-Branching problem. In this method we utilize the known polynomial kernel for the problem and obtain a collection of $2^{o(k)}$ instances such that the input instance is an "yes" instance if and only if one of the instances in our collection is. The property of the instances in the collection which we make use of is that the treewidth of the underlying undirected graph of these instances is $o(k)$. The last property brings dynamic programming on graphs of bounded treewidth into picture as the final step of the algorithm.

Here, we will solve a rooted version of the k-IOB problem, called Rooted k-Internal Out-Branching (R-k-IOB), where apart from D and k we are also given a root $r \in V(D)$, and the objective is to find an r-out-branching, if exists one, with at least k internal vertices. The k-IOB problem can be reduced to R-k-IOB by guessing the root r at the additional cost of $|V(D)|$ in the running time of the R-k-IOB problem. Henceforth, we will only consider R-k-IOB. We call an r-out-tree T with k internal vertices *minimal* if deleting any leaf results in an r-out-tree with at most $k-1$ internal vertices. A well known result relating minimal r-out-tree T with k internal vertices with a solution to R-k-IOB is as follows.

Lemma 4.1 ([9]). *Let D be a rooted connected digraph with root r. Then D has an r-out-branching T' with at least k internal vertices if and only if D has a minimal r-out-tree T with k internal vertices with $|V(T)| \le 2k-1$. Furthermore, given a minimal r-out-tree T, we can find an r-out-branching T' with at least k internal vertices in polynomial time.*

We also need another known result about kernelization for k-IOB.

Lemma 4.2 ([24]). *k-Internal Out-Branching admits a polynomial kernel of size $8k^2+6k$.*

In fact, the kernelization algorithm presented in [24] works for all digraphs and has a unique reduction rule which only *deletes vertices*. This implies that if we start with a graph $G \in \mathscr{G}$ where $\mathscr{G}$ excludes a fixed graph H as a minor, then the graph G' obtained after applying kernelization algorithm still belongs to $\mathscr{G}$.

Our algorithm tries to find a minimal r-out-tree T with k internal vertices with $|V(T)| \le 2k-1$ recursively. As the first step of the algorithm we obtain a set of $2^{o(k)}$ digraphs such that the underlying undirected graphs have treewidth $\mathcal{O}(\sqrt{k})$, and the original problem is a "yes" instance if and only at least one of the $2^{o(k)}$ instances is a "yes" instance. More formally, we prove the following lemma.

Lemma 4.3. [$\star$] *Let H be a fixed apex graph and $\mathscr{G}$ be a minor closed graph class excluding H as a minor. Let (D,k) be an instance to k-Internal Out-Branching such that $UG(D) \in \mathscr{G}$. Then there exists a collection*

$$\mathcal{C} = \left\{ (D_i, k', r) \mid D_i \text{ is a subgraph of } D, k' \le k, r \in V(D), 1 \le i \le \binom{8k^2+6k}{\sqrt{k}} \right\},$$

of instances such that $\mathbf{tw}(UG(D_i)) = \mathcal{O}(\sqrt{k})$ for all i and (D,k) has an out-branching with at least k internal vertices if and only if there exists an i, r and $k' \le k$ such that (D_i, k', r) has an r-out-branching with at least k' internal vertices.

Given a tree decomposition of width w for $UG(D)$, one can solve R-k-IOB in time $2^{\mathcal{O}(w \log w)}n$ using a dynamic programming over graphs of bounded treewidth as described in [24]. This brings us to the main theorem of this section.

Theorem 4.4. [$\star$] *The k-IOB problem can be solved in time $2^{\mathcal{O}(\sqrt{k} \log k)} + n^{\mathcal{O}(1)}$ on digraphs with n vertices such that the underlying undirected graph excludes a fixed apex graph H as a minor.*

5. Conclusion and Discussions

We have given the first subexponential parameterized algorithms on planar digraphs and on the class of digraphs whose underlying undirected graph excludes a fixed graph H or an apex graph as a minor. We have outlined two general techniques, and have illustrated them on two well studied problems concerning oriented spanning trees (out branching)—one that maximizes the number of leaves and the other that maximizes the number of internal vertices. One of our techniques uses the grid theorem on H-minor graphs, albeit in a different way than how it is used on undirected graphs. The other uses Baker type layering technique combined with kernelization and solves the problem on a subexponential number of problems whose instances have sublinear treewidth.

We believe that our techniques will be widely applicable and it would be interesting to find other problems where such subexponential algorithms are possible. Two famous open problems in this context are whether the k-Directed Path problem (does a digraph contains a directed path of length at least k) and the k-Directed Feedback Vertex Set problem (does a digraph can be turned into acyclic digraph by removing at most k vertices) have subexponential algorithms (at least) on planar digraphs. However, for the k-Directed Path problem, we can reach "almost" subexponential running time. More precisely, we have the following theorem.

Theorem 5.1. [$\star$] *For any $\varepsilon > 0$, there is δ such that the k-*Directed Path *problem is solvable in time $\mathcal{O}((1+\varepsilon)^k \cdot n^{\delta})$ on digraphs with n vertices such that the underlying undirected graph excludes a fixed apex graph H as a minor.*

Let use remark that similar $\mathcal{O}((1+\varepsilon)^k n^{f(\varepsilon)})$ results can also be obtained for many other problems including Planar Steiner Tree.

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] N. Alon, F. V. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Spanning directed trees with many leaves. *SIAM Journal on Discrete Mathematics*, 23(1):466–476, 2009.

[3] N. Alon, D. Lokshtanov, and S. Saurabh. Fast FAST. In *ICALP 09*, volume 5555 of *LNCS*, pages 49–58. Springer, 2009.

[4] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[5] B. S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

[6] P. Bonsma and F. Dorn. Tight bounds and a fast FPT algorithm for directed max-leaf spanning tree. *to appear in Transaction on Algorithms.*

[7] J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008.

[8] J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In *MFCS 06*, volume 4162 of *LNCS*, pages 238–249. Springer, 2006.

[9] N. Cohen, F. V. Fomin, G. Gutin, E. J. Kim, S. Saurabh, and A. Yeo. Algorithm for finding k-vertex out-trees and its application to k-internal out-branching problem. In *COCOON*, volume 5609 of *LNCS*, pages 37–46, 2009.

[10] J. Daligault, G. Gutin, E. J. Kim, and A. Yeo. FPT algorithms and kernels for the directed k-leaf problem. *CoRR*, abs/0810.4946, 2008.

[11] J. Daligault and S. Thomassé. On finding directed trees with many leaves. In *IWPEC 09*, LNCS, page to appear, Berlin, 2009. Springer-Verlag.

[12] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, 2005.

[13] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[14] E. D. Demaine and M. Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008.

[15] E. D. Demaine and M. T. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Computer Journal*, 51(3):292–302, 2008.

[16] E. D. Demaine, M. T. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *FOCS 05*, pages 637–646, 2005.

[17] F. Dorn, F. V. Fomin, and D. M. Thilikos. Catalan structures and dynamic programming in H-minor-free graphs. In *SODA 08*, pages 631–640. SIAM, 2008.

[18] F. Dorn, F. V. Fomin, and D. M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.

[19] F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *to appear in Algorithmica*.

[20] H. Fernau, F. V. Fomin, D. Lokshtanov, D. Raible, S. Saurabh, and Y. Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. In *STACS*, pages 421–432, 2009.

[21] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, Berlin, 2006.

[22] F. V. Fomin, S. Gaspers, S. Saurabh, and S. Thomassé. A linear vertex kernel for maximum internal spanning tree. In *ISAAC 09*, page to appear. Springer, 2009.

[23] F. V. Fomin and D. M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM J. Comput.*, 36:281–309, 2006.

[24] G. Gutin, E. J. Kim, and I. Razgon. Minimum leaf out-branching problems. *CoRR*, abs/0801.1979, 2008.

[25] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity. *J. Comput. System Sci.*, 63(4):512–530, 2001.

[26] J. Kneis, A. Langer, and P. Rossmanith. A new algorithm for finding trees with many leaves. In *ISAAC 08*, volume 5369 of *LNCS*, pages 270–281. Springer-Verlag, 2008.

[27] R. Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford University Press, Oxford, 2006.

[28] E. Prieto and C. Sloper. Reducing to independent set structure – the case of k-internal spanning tree. *Nord. J. Comput.*, 12(3):308–318, 2005.

[29] B. A. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

[30] N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Comb. Th. Ser. B*, 62:323–348, 1994.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 263-274
www.stacs-conf.org

PLANAR SUBGRAPH ISOMORPHISM REVISITED

FREDERIC DORN[1]

[1] Department of Informatics, University of Bergen, Norway
E-mail address: frederic.dorn@ii.uib.no

Abstract. The problem of Subgraph Isomorphism is defined as follows: Given a *pattern* H and a *host graph* G on n vertices, does G contain a subgraph that is isomorphic to H? Eppstein [SODA 95, J'GAA 99] gives the first linear time algorithm for subgraph isomorphism for a fixed-size pattern, say of order k, and arbitrary planar host graph, improving upon the $O(n^{\sqrt{k}})$-time algorithm when using the "Color-coding" technique of Alon et al [J'ACM 95]. Eppstein's algorithm runs in time $k^{O(k)}n$, that is, the dependency on k is superexponential. We improve the running time to $2^{O(k)}n$, that is, single exponential in k while keeping the term in n linear. Next to deciding subgraph isomorphism, we can construct a solution and count all solutions in the same asymptotic running time. We may enumerate ω subgraphs with an additive term $O(\omega k)$ in the running time of our algorithm. We introduce the technique of "embedded dynamic programming" on a suitably structured graph decomposition, which exploits the number and topology of the underlying drawings of the subgraph pattern (rather than of the host graph).

Introduction

In the literature, we often find results on polynomial time or even linear time algorithms for NP-hard problems. Take for example the NP-complete problem of computing an optimal tree-decomposition of a graph. Bodlaender [3] gives a linear time algorithm—restricted to graphs of constant treewidth. The Graph Minor Theory by Robertson and Seymour implies amongst others that there is an $O(n^3)$ algorithm for the disjoint path problem, that is for finding disjoint paths between a constant number of terminals. Taking a closer look at such results, one notices that a function exponential in size of some constant c is hidden in the O-notation of the running time—here, c is the treewidth and the number of terminals, respectively. In another line of research, *parameterized complexity*, the primary goal is to rather find algorithms that minimize the exponential term of the running time—the exponential function of the *problem parameter* k. The first step here is to prove that such an algorithm with a separate exponential function exists, that is, that the studied problem is *fixed parameter tractable (FPT)* [13, 16, 21]. Such problem has an algorithm with time complexity bounded by a function of the form $f(k) \cdot n^{O(1)}$, where the *parameter*

1998 ACM Subject Classification: F.2.2;G.2.1;G.2.2.

Key words and phrases: Graph algorithms; Subgraph Isomorphism; NP-hard problems; Dynamic programming; Topological graph theory.

Supported by the Research Council of Norway.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2460

function f is a computable function only depending on k. The second step in the design of FPT-algorithms is to decrease the growth rate of the parameter function.

We can identify two different trends in which exact algorithms are improved. Either one decreases the degree of the polynomial term in the asymptotic running time, or one focusses on obtaining parameter functions with better exponential growth. In the present work, we achieve both goals for the computational problem Planar Subgraph Isomorphism.

Subgraph Isomorphism generalizes many important graph problems, such as Hamiltonicity, Longest Path, and Clique. It is known to be NP-complete, even when restricted to planar graphs [18]. Until now, the best known algorithm to solve Subgraph Isomorphism, that is to find a subgraph of a given host graph isomorphic to a pattern H on k vertices, is the naïve exhaustive search algorithm with running time $O(n^k)$ and no FPT-algorithm can be expected here [13]. For a pattern H of treewidth at most t, Alon et al. [1] give an algorithm of running time $2^{O(k)}n^{O(t)}$. For Planar Subgraph Isomorphism, given planar pattern and input graph, some considerable improvements have been made mostly during the 90's ([23], [1]). The current benchmark has been set by Eppstein [14] to $k^{O(k)}n$, by employing graph decomposition methods, similar to the Baker-approach [2] for approximating NP-complete problems on planar graphs. Eppstein's algorithm is actually the first FPT-algorithm for Planar Subgraph Isomorphism with k as parameter. Eppstein poses three open problems: a) whether one can extend the technique in [1] to improve the dependence on the size of the pattern from $k^{O(k)}$ to $2^{O(k)}$ for the decision problem of subgraph isomorphism; and whether one can achieve similar improvements, b) for the counting version, and c) for the listing version of the subgraph isomorphism problem.

Our results. In this work, we do not only achieve this single exponential behavior in k for all three problems—without applying the randomized coloring technique—we also keep the term in n linear. That is, we give an algorithm for Planar Subgraph Isomorphism for a pattern H of order k with running time $2^{O(k)}n$. Next to deciding subgraph isomorphism, we can construct a solution and count all solutions in the same asymptotic running time. We may list ω subgraphs with an additive term $O(\omega k)$ in the running time of our algorithm. Our algorithm also improves the time complexity of [17] for large patterns of size $k \in o(\sqrt{n}\log n)$.

The novelty of our result comes from *embedded dynamic programming*, a technique we find interesting on its own. Here, one decomposes the graph by separating it into induced subgraphs. In the dynamic programming step, one computes partial solutions for the separated subgraphs, that are updated to an overall solution for the whole graph. In ordinary dynamic programming, one would argue how the subgraph pattern hits separators of the host graph. Instead, in embedded dynamic programming for subgraph isomorphism, we proceed exactly the opposite way: we look at how separators can be routed through the subgraph pattern. As a consequence, we bound the number of partial solutions by a function of both the separator size of the host graph and the pattern size—as it turns out, for the planar subgraph isomorphism problem, that function is single exponential in the number of vertices of the pattern. To obtain a good bound on the parameter function, we apply several fundamental enumerative combinatorics results in the technical sections of this work. Next to the number of face-vertex sequences in embedded graphs, these counting results give an upper bound on the number of planar drawings of the pattern.

Our algorithm is divided into two parts with the second part being the aforementioned embedded dynamic programming. For keeping the time complexity of our algorithm linear in the size of the host graph, we give a fast method for computing *sphere-cut decompositions*—natural extensions of tree-decompositions to plane graphs—with separators of size linearly bounded by the size of the subgraph pattern.

Theorem 0.1. *Let G be a planar graph on n vertices and H a pattern of order k. We can decide if there is a subgraph of G that is isomorphic to H in time $2^{O(k)}n$. We find subgraphs and count subgraphs of G isomorphic to H in time $2^{O(k)}n$ and enumerate ω subgraphs in time $2^{O(k)}n + O(\omega k)$.*

Let us mention that for k-Longest Path on planar graphs, the authors of [12] give the first algorithm with subexponential running time behaviour, namely $2^{O(\sqrt{k})}n + O(n^3)$, employing the techniques *Bidimensionality* and topological dynamic programming. *Bidimensionality Theory* employs results of Graph Minor Theory for planar graphs [24] and other structural graph classes to algorithmic graph theory (entry [6], for a survey [7]). Unfortunately, Bidimensionality does only work for finding specific patterns in a graph, such as k-paths, but not for subgraph isomorphism problems in general. For a survey on other planar subgraph isomorphism problems with restricted patterns, please consider [14].

Organization. Following the definitions in Section 1, we state in Section 2 how to obtain a sphere-cut decomposition of small width. In Section 3 we restrict Planar Subgraph Isomorphism to Plane Subdrawing Equivalence. We give some technical lemmas in Section 3.1 to bound the number of ways a separator of the sphere-cut decomposition can be routed through a plane pattern. We describe embedded dynamic programming in Section 3.2 and subsume the entire algorithm for Plane Subdrawing Equivalence in Section 3.3. In Section 4 we extend our algorithm for solving Planar Subgraph Isomorphism.

1. Preliminaries

Subgraph isomorphism. Let G, H be two graphs. We call G and H *isomorphic* if there exists a bijection $\nu : V(G) \to V(H)$ with $\{v, w\} \in E(G) \Leftrightarrow \{\nu(v), \nu(w)\} \in E(H)$. We call H *subgraph isomorphic to* G if there is a subgraph H' of G isomorphic to H.

Branch decompositions. A *branch decomposition* $\langle T, \mu \rangle$ of a graph G consists of an unrooted ternary tree T (internal vertex-degree 3) and a bijection $\mu : L \to E(G)$ from the set L of leaves of T to the edge set of G. We define for every edge e of T the *middle set* $\mathrm{mid}(e) \subseteq V(G)$ as follows: Let T_1 and T_2 be the two connected components of $T \setminus \{e\}$. Then let G_i be the graph induced by the edge set $\{\mu(f) : f \in L \cap V(T_i)\}$ for $i \in \{1, 2\}$. The *middle set* is the intersection of the vertex sets of G_1 and G_2, i.e., $\mathrm{mid}(e) := V(G_1) \cap V(G_2)$. The *width* bw of $\langle T, \mu \rangle$ is the maximum order of the middle sets over all edges of T, i.e., $\mathrm{bw}(\langle T, \mu \rangle) := \max\{|\,\mathrm{mid}(e)| : e \in T\}$. An optimal branch decomposition of G is defined by a tuple $\langle T, \mu \rangle$ which provides the minimum width, the *branchwidth* $\mathrm{bw}(G)$.

Plane graphs and equivalent drawings. Let Σ be the unit sphere. A *planar drawing* or simply *drawing* of a graph G with vertex set $V(G)$ and edge set $E(G)$ maps vertices to points in the sphere, and edges to simple curves between their end vertices, such that edges do not cross, except in common end vertices. A *plane graph* is a graph G together with a planar drawing. A *planar graph* is a graph that admits a planar drawing. For details, see e.g. [9]. The set of *faces* $F(G)$ of a plane graph G is defined as the union of the connected regions of $\Sigma \setminus G$. A subgraph of a plane graph G, induced by the vertices and edges incident to a face $f \in F(G)$, is called a *bound* of f. If G is 2-connected, each bound of a face is a cycle. We call this cycle *face-cycle* (for further reading, see e.g. [9]). For a subgraph H of a plane graph G, we refer to the drawing of G reduced to the vertices and edges of H as a *subdrawing* of G. Consider any two drawings G_1 and G_2 of a planar graph G. A *homeomorphism* of G_1 onto G_2 is a homeomorphism of Σ onto itself which maps vertices, edges, and faces of G_1 onto vertices, edges, and faces of G_2, respectively. We call two planar drawings *equivalent*, if there is a homeomorphism from one onto the other.

Theorem 1.1. *e.g.* [9] *Every 3-connected planar graph has a unique drawing in a sphere Σ up to homeomorphism.*

Proposition 1.2. *e.g.* [22] *Every planar n-vertex graph has $2^{O(n)}$ non-equivalent drawings.*

Remark 1.3. Let G and H be two plane graphs. If their drawings are equivalent, then G is isomorphic to H. On the contrary, if G is isomorphic to H and neither graphs are 3-connected, then their drawings are not necessarily equivalent.

Triangulations. We call a plane graph G a *planar triangulation* or simply a *triangulation* if every face in $F(G)$ is bounded by a triangle (a cycle of length three). If H is a subdrawing of a triangulation G, we call G a *triangulation of H*.

Nooses and combinatorial nooses. A *noose* of a Σ-plane graph G is a simple closed curve in Σ that meets G only in vertices. From the Jordan Curve Theorem, it then follows that nooses separate Σ into two regions. Let $V(N) = N \cap V(G)$ be the vertices and $F(N)$ be the faces intersected by a noose N. The *length* of N is the number $|V(N)|$ of vertices in $V(N)$. The clockwise order in which N meets the vertices of $V(N)$ is a cyclic permutation π on the set $V(N)$.

Remark 1.4. Let a plane graph H be a subdrawing of a plane graph G. Every noose N in G is also a noose in H and $N \cap V(H) \subseteq N \cap V(G)$.

A *combinatorial noose* $N_C = [v_0, f_0, v_1, f_1, \dots, f_{\ell-1}, v_\ell]$ in a plane graph G is an alternating sequence of vertices and faces of G, such that

- f_i is a face incident to both v_i, v_{i+1} for all $i < \ell$,
- $v_0 = v_\ell$ and the vertices $v_1, \dots, v_\ell$ are mutually distinct and
- if $f_i = f_j$ for any $i \neq j$ and $i, j = 0, \dots, \ell - 1$, then the vertices v_i, v_{i+1}, v_j, and v_{j+1} do not appear in the order $(v_i, v_j, v_{i+1}, v_{j+1})$ on the bound of face $f_i = f_j$.

The *length* of a combinatorial noose $[v_0, f_0, v_1, f_1, \dots, f_{\ell-1}, v_\ell]$ is ℓ.

Remark 1.5. The order in which a noose N intersects the faces $F(N)$ and the vertices $V(N)$ of a plane graph G gives a unique alternating face-vertex sequence of $F(N) \cup V(N)$ which is a combinatorial noose N_C. Conversely, for every combinatorial noose N_C there exists a noose N with face-vertex sequence N_C.

We may view combinatorial nooses as equivalence classes of nooses, that can be represented by the same face-vertex sequence.

Sphere cut decompositions. For a Σ-plane graph G, we define a *sphere cut decomposition* or *sc-decomposition* $\langle T, \mu, \pi \rangle$ as a branch decomposition which for every edge e of T has a noose N_e that cuts Σ into two regions Δ_1 and Δ_2 such that $G_i \subseteq \Delta_i \cup N_e$, where G_i is the graph induced by the edge set $\{\mu(f) : f \in L \cap V(T_i)\}$ for $i \in \{1, 2\}$ and $T_1 \dot\cup T_2 = T \setminus \{e\}$. Thus N_e meets G only in $V(N_e) = \operatorname{mid}(e)$ and its length is $|\operatorname{mid}(e)|$. The vertices of $\operatorname{mid}(e) = V(G_1) \cap V(G_2)$ are enumerated according to a cyclic permutation π on $\operatorname{mid}(e)$.

The following two propositions will be crucial in that they give us upper bounds on the number of partial solutions we will compute in our dynamic programming approach. With both propositions, we will bound the number of combinatorial nooses in a plane graph by the number of cycles in the triangulation of some auxiliary graph.

Proposition 1.6. *(*[4]*) No planar n-vertex graph has more than $2^{1.53n}$ simple cycles.*

Proposition 1.7. *(*[27]*) The number of non-isomorphic maximal planar graphs on n vertices is approximately $2^{3.24n}$.*

Proposition 1.7 also gives a bound on the number of non-isomorphic triangulations. Any drawing of a maximal planar graph G must be a triangulation, otherwise G would not be maximal. With Theorem 1.1, every maximal planar graph has a unique drawing which is a triangulation. On the other hand, every triangulated graph is maximal planar.

2. Computing sphere-cut decompositions in linear time

In this section we sketch an algorithm for computing sc-decompositions of bounded width. Let H be a connected subgraph of G with $|V(H)| = k$, and let $v \in V(H)$. Then H is a subgraph of the induced subgraph G^v of G, where $G^v = G[S]$ with $S = \{w \in S \mid \text{dist}(v, w) \leq k\}$ ($\text{dist}(v, w)$ denotes the length of a shortest path between v and w in G). This observation helps us to shrink the search space of our algorithm by cutting out chunks of G of bounded width and solve subgraph isomorphism separately on each chunk. With the algorithm of Tamaki [26], one can compute a branch decomposition of G^v of width $\leq 2k + 1$, following similar ideas as in the approach of Baker [2] for tree decompositions. With some simple modifications, we achieve the same result for sc-decompositions. In an extended version of this paper [10], we prove the following lemma and give an algorithm that computes a sc-decomposition of bounded width in linear time.

Lemma 2.1. *(*[2],[26],[10]*) Let G be a plane graph with a rooted spanning tree whose root-leaf-paths have length $\leq k$. We can find an sc-decomposition of width $2k+1$ in time $O(kn)$.*

3. Plane Subdrawing Equivalence

In this section, we study the variant of the subgraph isomorphism problem on patterns and host graphs drawn in the unit sphere. In Plane Subdrawing Equivalence, the question is to find a subdrawing of a plane host graph G that is equivalent to the drawing of a plane pattern H. By Remark 1.3, the problem is equivalent to Planar Subgraph Isomorphism for 3-connected planar graphs. In Section 4 we carry over our results to all planar graphs. We first introduce some topological tools that we need for embedded dynamic programming. At every step of the dynamic programming, we compute every way how a combinatorial noose N corresponding to a middle set of the sc-decomposition $\langle T, \mu, \pi \rangle$ of G can intersect a subdrawing equivalent to the drawing of pattern H. Each intersection gives rise to a combinatorial noose of H. See Figure 1 for an illustration.

The running time of the algorithm crucially depends on the number of combinatorial nooses in H. The aim of this section is to prove the following:

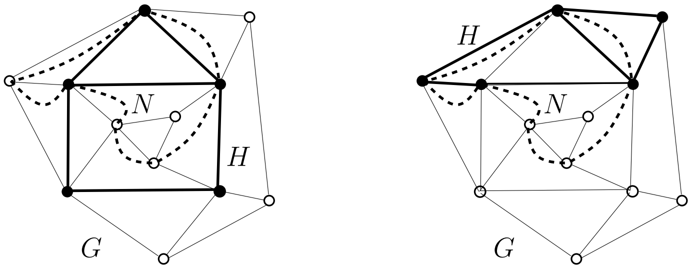


Figure 1: On the left, we draw graph G with an emphasized subdrawing H intersected by a combinatorial noose N indicated by dashed lines. On the right, we have the same graph G with a different copy of H intersected by N.

Theorem 3.1. *Let G be a plane graph on n vertices and H be a plane graph on $k \leq n$ vertices. We can decide if there is a subdrawing of G that is equivalent to the drawing of H in time $2^{O(k)}n$. We can find and count subdrawings equivalent to the drawing of H in time $2^{O(k)}n$, and enumerate ω subdrawings in time $2^{O(k)}n + O(\omega k)$.*

3.1. Combinatorial nooses in plane graphs

For a refined algorithm analysis we now take a close look at combinatorial nooses of plane graphs. In particular we are interested in counting the number of combinatorial nooses. In this subsection, we will prove the following proposition:

Proposition 3.2. *Every plane k-vertex graph has $2^{O(k)}$ combinatorial nooses.*

Before proving this proposition, we state that every combinatorial noose of a plane graph on k vertices corresponds to a cycle in some other plane graph on at most $O(k)$ vertices. The proofs of the following lemmas can be found in [10]. First we relate combinatorial nooses in a planar triangulation H to the cycles of H. Then we state that for any plane graph H there is an auxiliary graph H^*, such that the combinatorial nooses of H can be injectively mapped to the cycles of the triangulations of H^*. From Proposition 1.6 we know an upper bound on the number of cycles in planar graphs, which we employ to prove Proposition 3.2.

Lemma 3.3. *Let H be a planar triangulation and $N_C = [v_0, f_0, v_1, f_1, \ldots, f_{\ell-1}, v_\ell]$ a combinatorial noose of H. Then for every pair of consecutive vertices v_i, v_{i+1} in N_C, there is a unique edge $\{v_i, v_{i+1}\}$ in $E(H)$. That is, the sequence $[v_0, v_1, \ldots, v_\ell]$ is a simple cycle in H if $|V(N_C)| > 2$, and if $|V(N_C)| = 2$, it corresponds to a single edge in H.*

For an edge $e = \{v, w\}$ of a graph H we *subdivide* e by adding a vertex u to $V(H)$ and replacing e by two new edges $e_1 = \{v, u\}$ and $e_2 = \{u, w\}$. In a drawing of H, we place point u in the middle of the drawing of e partitioning e into e_1 and e_2.

Lemma 3.4. *Let H be plane graph and $N_C = [v_0, f_0, v_1, f_1, \ldots, f_{\ell-1}, v_\ell]$ a combinatorial noose of H with $|V(N_C)| > 2$. Let H^* be obtained by subdividing every edge in $E(H)$. There exists a planar triangulation H' of H^* such that $[v_0, v_1, \ldots, v_\ell]$ is a cycle in H'.*

Proof of Proposition 3.2. If H is triangulated, we have with Lemma 3.3 that every combinatorial noose corresponds to a unique cycle in H. By Proposition 1.6, the number of cycles in H is bounded by $2^{1.53k}$. Since for every edge of a cycle in H, we have two choices for a combinatorial noose to visit an incident face, we get the overall upper bound of $2^{2.53k}$ on the number of combinatorial nooses. If H is plane, we have to count the triangulations of H^* (Lemma 3.4). By Proposition 1.7 and the comments below it, there are at most $2^{3.24n}$ non-isomorphic triangulations on n vertices. Let us denote this set of triangulated graphs by Φ. We note that H^* is a subgraph of some graph of Φ, say of all graphs in $\Phi_H \subseteq \Phi$ with $|\Phi_H| \geq 1$. Since every triangulated graph is 3-connected, we have with Theorem 1.1 that every graph H' in Φ_H has a unique drawing in Σ up to homeomorphism. The plane graph H^* is then a subdrawing of a drawing equivalent to an arbitrary planar drawing of H' in Σ. The number of triangulations times the number of combinatorial nooses in each triangulation is an upper bound on the number of combinatorial nooses in H^*. ■

For embedded dynamic programming on a sc-decomposition $\langle T, \mu, \pi \rangle$, we can argue with Remark 1.4 that if H is a subdrawing of G, then noose N formed by the middle set mid(e) is a noose of H, too. Recalling Remark 1.5, the alternating sequence of vertices and faces of H visited by N forms a combinatorial noose N_C in H. This observation allows us to discuss the results from a combinatorial point of view without the underlying topological arguments. Instead of nooses we will refer to combinatorial nooses in the remaining section.

3.2. Embedded dynamic programming

In embedded dynamic programming, the basic difference to usual dynamic programming is that we do not check for every partial solution for a given problem if or how it lies in the graph processed so far. Instead, we check how the graph that we have processed so far is intersecting the entire solution, that is how the graph is *embedded* into our solution. For subdrawing equivalence, we are interested in how G is drawn in the plane pattern H up to homeomorphism. Each edge of an sc-decomposition tree T corresponds to a noose N of G. We will associate to N the list of all possible subgraphs of H that appear in the part of G bounded by N. Therefore, we will describe all possible ways H is intersected by N. The number of solutions we get is bounded by the number of combinatorial nooses in H we can map N onto. We describe the algorithm in what follows.

Dynamic programming. We root sc-decomposition $\langle T, \mu, \pi \rangle$ at some node $r \in V(T)$. For each edge $e \in T$, let L_e be the set of leaves of the subtree rooted at e. The subgraph G_e of G is induced by the edge set $\{\mu(v) \mid v \in L_e\}$. The vertices of mid$(e)$ form a combinatorial noose N that separates G_e from the residual graph.

Assuming H is a subgraph of G, the basic idea of embedded dynamic programming is that we are interested in how the vertices of the combinatorial noose N are intersecting faces and vertices of H. Since every noose in G is a noose in H, we can map N to a combinatorial noose N^H of H, bounding (clockwise) a unique subgraph H_{sub} of H.

In each step of the algorithm, all solutions for a sub-problem in G_e are computed, namely all possibilities of how N is mapped onto a combinatorial noose N^H in H that separates H_{sub} from the rest of H, where $H_{sub} \subseteq H$ is isomorphic to subgraphs of G_e. For every middle set, we store this information in an array. It is updated in a bottom-up process starting at the leaves of $\langle T, \mu, \pi \rangle$. During this updating process it is guaranteed that the 'local' solutions for each subgraph associated with a middle set of the sc-decomposition are combined into a 'global' solution for the overall graph G.

Valid mappings. Let G be a plane graph with a rooted sc-decomposition $\langle T, \mu, \pi \rangle$ and let H be a plane pattern. For every middle set $\text{mid}(e)$ of $\langle T, \mu, \pi \rangle$ let N be the associated combinatorial noose in G with face-vertex sequence of $F(N) \cup V(N)$. Let $\mathfrak{L}$ denote the set of all combinatorial nooses of H whose length is at most the length of N. We now want to map N order preserving to each $N^H \in \mathfrak{L}$. We map vertices of N to both vertices and faces of H. Therefore, we consider partitions of $V(N) = V_1(N) \dot{\cup} V_2(N)$ where vertices in $V_1(N)$ are mapped to vertices of $V(H)$ and vertices in $V_2(N)$ to faces of $F(H)$. We define a mapping $\gamma : V(N) \cup F(N) \to V(H) \cup F(H)$ relating N to the combinatorial nooses in $\mathfrak{L}$. For every $N^H \in \mathfrak{L}$ on faces and vertices of set $F(N^H) \cup V(N^H)$ and for every partition $V_1(N) \dot{\cup} V_2(N)$ of $V(N)$ mapping γ *is valid* if

a) γ restricted to $V_1(N)$ is a bijection to $V(N^H)$;

b) for every $v \in V_2(N)$ and $f \in F(N)$ we have $\gamma(v)$ and $\gamma(f)$ in $F(N^H)$;

c) for every $v_i \in V(N)$ and subsequence $[f_{i-1}, v_i, f_i]$ of N, face $\gamma(v_i)$ is equal to both $\gamma(f_{i-1})$ and $\gamma(f_i)$, and vertex $\gamma(v_i)$ is incident to both $\gamma(f_{i-1})$ and $\gamma(f_i)$;

d) for every pair $w_i, w_j \in V(N^H)$: if $\{w_i, w_j\} \in E(H)$ then $\{\gamma^{-1}(w_i), \gamma^{-1}(w_j)\} \in E(G)$.

Items *a*) and *b*) say where to map the faces and vertices of N to. Item *c*) (with *a*)) makes sure that if two vertices v_h, v_j in sequence $N = [\ldots, v_h, \underline{\ldots}, v_j, \ldots]$ are mapped to two vertices w_i, w_{i+1} that appear in sequence N^H as $[\ldots, w_i, f_i, w_{i+1}, \ldots]$ then every face and vertex inbetween v_h, v_j in sequence N (here underlined) is mapped to face f_i. Item *d*) rules out the invalid solutions, that is, we do not map a pair of vertices in G that have no edge in common to the endpoints of an edge in H. We do so because if H is a subgraph of G then an edge in H is an edge in G, too. For an illustration, see Figure 2.

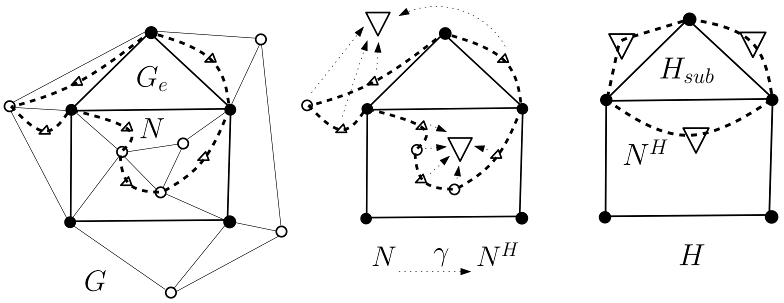


Figure 2: On the left, we have a plane graph G with a subgraph H emphasized. A combinatorial noose N separating subgraph G_e is indicated by dashed lines. The vertices of N are full and empty circles and the faces triangles. In the middle, we have H and indicate to which faces (big triangles) of H vertices and faces of N are mapped by γ. This gives us combinatorial noose N^H on the right, separating subgraph H_{sub}.

We assign an array A_e to each $\text{mid}(e)$ consisting of all tuples $\langle N^H, \gamma_e \rangle$ each representing a valid mapping γ_e from combinatorial noose N corresponding to $\text{mid}(e)$ to a combinatorial noose $N^H \in \mathfrak{L}$. The vertices and faces of N are oriented clockwise around the drawing of G_e. Without loss of generality, we assume for every $\langle N^H, \gamma_e \rangle \in A_e$ the orientation of N^H to be clockwise around the subdrawing H_{sub} of H equivalent to a subdrawing of G_e.

Step 0: Initializing the leaf edges. For each parent edge e_ℓ of a leaf ℓ of T we initialize the valid mappings from the combinatorial noose bounding the edge $\mu(\ell)$ of G to every combinatorial noose in H of length at most two.

Step 1: Update process. We update the arrays of the middle sets in post-order manner from the leaves of T to root r. In each dynamic programming step, we compare the arrays of two middle sets $\mathrm{mid}(e), \mathrm{mid}(f)$ in order to create a new array assigned to the middle set $\mathrm{mid}(g)$, where e, f and g have a vertex of T in common. From [12] we know about a special property of sc-decompositions: namely that the combinatorial noose N_g is formed by the symmetric difference of the combinatorial nooses N_e, N_f and that $G_g = G_e \cup G_f$. In other words, we are ensured that if two solutions on G_e and G_f bounded by N_e and N_f *fit together*, then they form a new solution on G_g bounded by N_g. We now determine when two solutions represented as tuples in the arrays A_e and A_f fit together. We update two tuples $\langle N_e^H, \gamma_e \rangle \in A_e$ and $\langle N_f^H, \gamma_f \rangle \in A_f$ to a new tuple in A_g if

- for every $x \in (\, V(N_e) \cup F(N_e)\,) \cap (\, V(N_f) \cup F(N_f)\,)$, we have $\gamma_e(x) = \gamma_f(x)$;
- for the subgraph H_e of H separated by N_e^H and the subgraph H_f of H separated by N_f^H, we have that $E(H_e) \cap E(H_f) = \emptyset$ and $V(H_e) \cap V(H_f) \subseteq \{\gamma(v) \mid v \in V(N_e) \cap V(N_f)\}$.

If N_e and N_f fit together, we get a valid mapping $\gamma_g : N_g \to N_g^H$ as follows:

- for every $x \in (\, V(N_e) \cup F(N_e)\,) \cap (\, V(N_f) \cup F(N_f)\,) \cap (\, V(N_g) \cup F(N_g),\,)$ we have $\gamma_e(x) = \gamma_f(x) = \gamma_g(x)$;
- for every $y \in (\, V(N_e) \cup F(N_e)\,) \setminus (\, V(N_f) \cup F(N_f)\,)$ we have $\gamma_e(y) = \gamma_g(y)$;
- for every $z \in (\, V(N_f) \cup F(N_f)\,) \setminus (\, V(N_e) \cup F(N_e)\,)$ we have $\gamma_f(z) = \gamma_g(z)$.

We have that γ_g is a valid mapping from N_g to the combinatorial noose N_g^H that bounds subgraph $H_g = H_e \cup H_f$. Thus, we add tuple $\langle N_g^H, \gamma_g \rangle$ to array A_g.

Step 2: End of DP. If, at some step, we have a solution where the entire subgraph H is formed, we exit the algorithm confirming. That is, if $H = H_e \cup H_f$ and H_i is bounded by N_i (for both $i \in \{e, f\}$) then the combinatorial noose N_g is bounding the subdrawing of G equivalent to the drawing of H. We output this subdrawing by reconstructing the solution top-down in $\langle T, \mu, \pi \rangle$. If at root r no subdrawing equivalent to the drawing of H has been found, we output 'FALSE'.

Correctness of DP. Let plane graph H be a subdrawing of G. We have already seen how to map every combinatorial noose of G that identifies a separation of G via a valid mapping γ to a combinatorial noose of H determining a separation of H. Step 0 ensures that every edge of H is bounded by a combinatorial noose N^H of length two, which is determined by tuple $\langle N^H, \gamma \rangle$ in an array assigned to a leaf edge of T. We need to show that Step 1 computes a valid solution for N_g from N_e and N_f for incident edges e, f, g. We note that the property that the symmetric difference of the combinatorial nooses N_e and N_f forms a new combinatorial noose N_g is passed on to the combinatorial nooses N_e^H, N_f^H and N_g^H of H, too. If the two solutions fit together, then H_e of H separated by N_e^H and subgraph H_f of H separated by N_f^H only intersect in the image of $V(N_e) \cap V(N_f)$. We may observe that N_e^H and N_f^H intersect in a continuous alternating subsequence with order reversed to each other, i.e., $N_e^H \mid_{N_e \cap N_f} = \overline{N_f^H} \mid_{N_e \cap N_f}$, where $\overline{N^H}$ means the reversed sequence N^H. Since every oriented N^H identifies uniquely a separation of $E(H)$, we can easily determine if two tuples $\langle N_e^H, \gamma_e \rangle \in A_e$ and $\langle N_f^H, \gamma_f \rangle \in A_f$ fit together and form a new subgraph of H. If H is a subdrawing of G, then at some step we will enter Step 2 and produce the entire H.

Running time analysis. We first give an upper bound on the size of each array. The number of combinatorial nooses in $\mathfrak{L}$ we are considering is bounded by the total number of combinatorial nooses in H, which is $2^{O(|V(H)|)}$ by Proposition 3.2. The number of partitions of vertices of any combinatorial noose N is bounded by $2^{|V(N)|}$. Since the order of both N^H and N is given we only have $2|V(H)|$ possibilities to map vertices of N to N^H, once the vertices of N are partitioned. Thus, in an array A_e we may have up to $2^{O(|V(H)|)} \cdot 2^{|V(N)|} \cdot |V(H)|$ tuples $\langle N_e^H, \gamma_e \rangle$. We first create all tuples in the arrays assigned to the leaves. Since middle sets of leaves only consist of an edge in G, we get arrays of size $O(|V(H)|^2)$ which we compute in the same asymptotic running time. When updating middle sets $\mathrm{mid}(e), \mathrm{mid}(f)$, we compare every tuple of one array A_e to every tuple in array A_f to check if two tuples fit together. We can compute the unique subgraph H_e (resp. H_f) described by a tuple in A_e (resp. A_f), compare two tuples in A_e, A_f and create a new tuple in A_g in time linear in the order of $V(N)$ and $V(H)$. Since the size of A_g is bounded by $2^{O(|V(H)|)} \cdot 2^{O(|V(N)|)}$, the update process for two middle sets takes the same asymptotic time. Assuming sc-decomposition $\langle T, \mu, \pi \rangle$ of G has width ω and $|V(H)| \leq \omega$, we get the following result.

Lemma 3.5. *For a plane graph G with a given sc-decomposition $\langle T, \mu, \pi \rangle$ of G of width w and a plane pattern H on $k \leq w$ vertices we can search for a subdrawing of G equivalent to H in time $2^{O(w)} \cdot n$.*

3.3. The algorithm

We present the overall algorithm for solving Plane Subdrawing Equivalence with running time stated in Theorem 3.1.

Algorithm 3.1: Plane Subdrawing Equivalence: PLSE.

```
Input   : Plane graph $G$; Plane pattern $H$ of order $k$.
1 Choose an arbitrary vertex $v$ in $G$.
2 Partition $V(G)$ into $S_0 \cup S_1 \cup \ldots \cup S_\ell$ with $S_i = \{w \in V(G) : \mathrm{dist}(v, w) = i\}$
3 for every $G_i = G[S_i \cup \ldots \cup S_{i+k}]$ with $0 \leq i \leq \ell - k$ do
4     Compute sc-decomposition $\langle T, \mu, \pi \rangle$ of $G_i$.
5     Do embedded dynamic programming on $\langle T, \mu, \pi \rangle$ to find a subdrawing of $G_i$
      equivalent to the drawing of $H$ and intersecting $S_i$.
```

Partitioning the vertex set in Line 2 of Algorithm 3.1 PLSE, is a similar approach to the well-known Baker-approach [2]. Every vertex set S_i contains the vertices of distance i to the chosen vertex v. $S_0 = \{v\}$ and ℓ is the maximum distance in G from v. The graph G_i in Line 3 is induced by the sets $S_i, \ldots, S_{i+k}$. As in [14], we may argue that every vertex in G appears in at most k subgraphs G_i. This keeps our running time linear in n. We can apply Lemma 2.1 to each G_i in Line 4 to a compute sc-decomposition $\langle T, \mu, \pi \rangle$ of width $\leq 2k+1$, by adding a root vertex r for the BFS tree and make r adjacent to every vertex in S_i. The dynamic programming approach can easily be turned into an algorithm counting subdrawing equivalences (similar to [14]), by using a counter in the dynamic programming. Using an inductive argument, for every subgraphs G_i in Line 5 we only compute subgraphs intersecting with vertices in S_i and thus omit double-counting. We can adopt our technique to list the subdrawings of G equivalent to the drawing of H.

4. Planar subgraph isomorphism

Now we consider the case when both pattern H and host graph G are planar but not plane. From Remark 1.3 we know that two isomorphic planar graphs must not need to come with equivalent drawings. However, we observe that if H is isomorphic to a subgraph of G, then for every planar drawing of G there exists a drawing of H that is equivalent to a subdrawing of G. Hence, we may simply draw G planarly, and run the algorithm of the previous section for all non-equivalent drawings of H.

Algorithm 4.1: Planar subgraph isomorphism.

```
Input   : Planar graph $G$, Planar pattern $H$ of size $k$.
Compute a planar drawing of $G$.
if $H$ 3-connected then Return PLSE($G, H$).
for every non-equivalent drawing $I$ of $H$ do
    Return PLSE($G, I$).
```

The whole algorithm. We compute in Algorithm 4.1 every non-equivalent drawing of H as follows. First, we compute the set $\mathcal{H}$ of non-isomorphic maximal planar graphs in time proportional to its size using the algorithm in [20]. For every graph $H' \in \mathcal{H}$ and every subdrawing I of H' we check whether I is isomorphic to H by using the linear time algorithm for planar graph isomorphism in [19][1] . By Proposition 1.2, we then call Algorithm 3.1 $2^{O(k)}$ times, for each plane graph I isomorphic to H. This ensures us that Algorithm 3.1 has running time as stated in Theorem 0.1 [2].

Conclusion

We have shown how to use topological graph theory to improve the results on the already mentioned variations of Planar Subgraph Isomorphism, solving the open problems posed in [14] and [12]. With the results of [15], [14] extends the feasible graph class from planar graphs to apex-minor-free graphs. This cannot be done with the tools presented here. However, the authors of [11] devise a truly subexponential algorithm for k-Longest Path in H-minor-free graphs and thus apex-minor-free graphs, employing the structural theorem of Robertson and Seymour [25] and the results of [8, 5]. Can the structure of H-minor-free graphs, be exploited for our purposes?

It seems unlikely that our work can be extended to obtain a subexponential algorithm. The first reason, mentioned in the introduction, is that Bidimensionality applies to subgraphs with minor properties rather than to general subgraphs. Secondly, our enumerative bounds are either tight or of lower bound $2^{\Omega(k)}$. We want to pose the open problem: Is Plane Subdrawing Equivalence solvable in time $2^{o(k)}n^{O(1)}$?

Acknowledgments. The author thanks Paul Bonsma, Holger Dell and Fedor Fomin for discussions and comments of great value to the presentation of these results.

[1] We get a list of drawings of H, from which we can delete equivalent drawings by a modification of the algorithm in [19]—namely isomorphism test for face-vertex graphs.

[2] It can be show that Algorithm 3.1 runs in time $O(2^{12.57k}n)$ and Algorithm 4.1 in $O(2^{18.81k}n)$

References

[1] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. Assoc. Comput. Mach., 42 (1995), pp. 844–856.

[2] B. S. Baker, *Approximation algorithms for NP-complete problems on planar graphs*, J. Assoc. Comput. Mach., 41 (1994), pp. 153–180.

[3] H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.

[4] K. Buchin, C. Knauer, K. Kriegel, A. Schulz, and R. Seidel, *On the number of cycles in planar graphs*, in Proc. of the 13th Annual International Conference on Computing and Combinatorics (COCOON'07), vol. 4598 of LNCS, Springer, 2007, pp. 97–107.

[5] A. Dawar, M. Grohe, and S. Kreutzer, *Locally excluding a minor*, in Proc. of the 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), IEEE Computer Society, 2007, pp. 270–279.

[6] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos, *Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs*, J. ACM, 52 (2005), pp. 866–893.

[7] E. D. Demaine and M. T. Hajiaghayi, *The bidimensionality theory and its algorithmic applications*, Computer J., 51 (2008), pp. 292–302.

[8] E. D. Demaine, M. T. Hajiaghayi, and K. Kawarabayashi, *Algorithmic graph minor theory: Decomposition, approximation, and coloring*, in Proc. of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), IEEE Computer Society, 2005, pp. 637–646.

[9] R. Diestel, *Graph theory*, vol. 173 of Grad. Texts in Math., Springer, New York, third ed., 2000.

[10] F. Dorn, *Planar Subgraph Isomorphism Revisited*, http://arxiv.org/abs/0909.4692, 2009.

[11] F. Dorn, F. V. Fomin, and D. M. Thilikos, *Catalan structures and dynamic programming on H-minor-free graphs*, in Proc. of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008), ACM, New York, 2008, pp. 631–640.

[12] F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin, *Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions*, Algorithmica, (2009, to appear).

[13] R. G. Downey and M. R. Fellows, *Parameterized complexity*, Springer, New York, 1999.

[14] D. Eppstein, *Subgraph isomorphism in planar graphs and related problems*, J. Graph Alg. and Appl., 3 (1999), pp. 1–27.

[15] D. Eppstein, *Diameter and treewidth in minor-closed graph families*, Algorithmica, (2009).

[16] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Texts in Theoretical Computer Science. EATCS Series, Springer, Berlin, 2006.

[17] F. V. Fomin and D. M. Thilikos, *New upper bounds on the decomposability of planar graphs*, J. Graph Theory, 51 (2006), pp. 53–81.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[19] J. E. Hopcroft and J. K. Wong, *Linear time algorithm for isomorphism of planar graphs (preliminary report)*, in Proc. of the Sixth Annual ACM Symposium on Theory of Computing (STOC'74), ACM, 1974, pp. 172–184.

[20] Z. Li and S.-I. Nakano, *Efficient generation of plane triangulations without repetitions*, in Proc. of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01), vol. 2076 of LNCS, Springer, 2001, pp. 433–443.

[21] R. Niedermeier, *Invitation to fixed-parameter algorithms*, vol. 31 of Oxford Lecture Series in Mathematics and its Applications, Oxford University Press, Oxford, 2006.

[22] D. Osthus, H. J. Prömel, and A. Taraz, *On random planar graphs, the number of planar graphs and their triangulations*, J. Combin. Theory Ser. B, 88(1) (2003), pp. 119–134.

[23] J. Plehn and B. Voigt, *Finding minimally weighted subgraphs*, in Proc. of the 16th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'90), vol. 484 of LNCS, Springer, 1990, pp. 18–29.

[24] N. Robertson, P. Seymour, and R. Thomas, *Quickly excluding a planar graph*, J. Combin. Theory Ser. B, 62 (1994), pp. 323–348.

[25] N. Robertson and P. D. Seymour, *Graph minors. XVI. Excluding a non-planar graph*, J. Combin. Theory Ser. B, 89 (2003), pp. 43–76.

[26] H. Tamaki, *A linear time heuristic for the branch-decomposition of planar graphs*, in Proc. of the 11th Annual European Symposium on Algorithms (ESA'03), vol. 2832 of LNCS, Springer, 2003, pp. 765–775.

[27] W. T. Tutte, *A census of planar triangulations*, Canad. J. Math., 14 (1962), pp. 21–38.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 275-286
www.stacs-conf.org

INTRINSIC UNIVERSALITY IN SELF-ASSEMBLY

DAVID DOTY[1] AND JACK H. LUTZ[2] AND MATTHEW J. PATITZ[2] AND SCOTT M. SUMMERS[2] AND DAMIEN WOODS[3]

[1] Department of Computer Science, University of Western Ontario
London, Ontario, Canada, N6A5B7.
E-mail address, David Doty: ddoty@csd.uwo.ca

[2] Department of Computer Science, Iowa State University
Ames, IA 50011 USA.
E-mail address, Jack H. Lutz: lutz@cs.iastate.edu
E-mail address, Matthew J. Patitz: patitz@cs.iastate.edu
E-mail address, Scott M. Summers: summers@cs.iastate.edu

[3] California Institute of Technology, Pasadena, CA 91125, USA.
E-mail address: woods@caltech.edu

Abstract. We show that the Tile Assembly Model exhibits a strong notion of universality where the goal is to give a single tile assembly system that simulates the behavior of any other tile assembly system. We give a tile assembly system that is capable of simulating a very wide class of tile systems, including itself. Specifically, we give a tile set that simulates the assembly of any tile assembly system in a class of systems that we call *locally consistent*: each tile binds with exactly the strength needed to stay attached, and that there are no glue mismatches between tiles in any produced assembly.

Our construction is reminiscent of the studies of *intrinsic universality* of cellular automata by Ollinger and others, in the sense that our simulation of a tile system T by a tile system U represents each tile in an assembly produced by T by a $c \times c$ block of tiles in U, where c is a constant depending on T but not on the size of the assembly T produces (which may in fact be infinite). Also, our construction improves on earlier simulations of tile assembly systems by other tile assembly systems (in particular, those of Soloveichik and Winfree, and of Demaine et al.) in that we simulate the actual process of self-assembly, not just the end result, as in Soloveichik and Winfree's construction, and we do not discriminate against infinite structures. Both previous results simulate only temperature 1 systems, whereas our construction simulates tile assembly systems operating at temperature 2.

1998 ACM Subject Classification: Theory.

Key words and phrases: Biomolecular computation, intrinsic universality, self-assembly.

This research was supported in part by National Science Foundation Grants 0652569, 0728806 and 0832824, by the Spanish Ministry of Education and Science (MEC) and the European Regional Development Fund (ERDF) under project TIN2005-08832-C03-02, by Junta de Andalucía grant TIC-581, and by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant R2824A01 and the Canada Research Chair Award in Biocomputing.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2461

1. Introduction

The development of DNA tile self-assembly has moved nanotechnology closer to the goal of engineering useful systems that assemble themselves from molecular components. Since Seeman's pioneering work in the 1980s [21], many laboratory experiments have shown that DNA tiles can be designed to spontaneously assemble with one another into desired structures [20]. As physical and mathematical error-suppression techniques improve [3, 8, 13, 22, 24], this molecular programming of matter will become practical at ever larger scales.

The Tile Assembly Model, developed by Winfree [19, 26], is a discrete mathematical model of DNA tile self-assembly that enables us to explore the potentialities and limitations of this kind of molecular programming. It is essentially an "effectivization" of classical Wang tiling [25] in which the fundamental components are un-rotatable, but translatable square "tile types" whose sides are labeled with glue "colors" and "strengths." Two tiles that are placed next to each other *interact* if the glue colors on their abutting sides match, and they *bind* if the strength on their abutting sides matches with total strength at least a certain ambient "temperature." Extensive refinements of the abstract Tile Assembly Model were given by Rothemund and Winfree in [18, 19]. (Consult the technical appendix for full details of the abstract Tile Assembly Model.) The model deliberately oversimplifies the physical realities of self-assembly, but Winfree proved that it is Turing universal [26], implying that self-assembly can be algorithmically directed.

In this paper we investigate whether the Tile Assembly Model is capable of a much stronger notion of universality where the goal is to give a single tile assembly system that simulates the behavior of any other tile assembly system. We give a tile assembly system that is capable of simulating a very wide class of tile systems, including itself. Our notion of simulation is inspired by, but somewhat stronger than, intrinsic universality in cellular automata [2, 7, 14–16]. In our construction a simulated tile assembly system is encoded in a seed assembly of the simulating system. This encoding is done in a very simple (logspace computable) way. The seed assembly then grows to form an assembly that is a re-scaled (larger) version of the simulated assembly, where each tile in the latter is represented by a supertile (square of tiles) in the simulator. Not only this, but each of the possible (nondeterministically chosen) assembly sequences of the simulated tile system is modeled by a possible assembly sequence in the simulating system (also nondeterministically chosen). The latter property of our system is important and highlights one way in which this work distinguishes itself from other notions of intrinsic universality found in the cellular automata literature: not only do we want to simulate the final assembly but we also want the simulator to have the ability to dynamically simulate each of the valid growth processes that could lead to that final assembly.

A second distinguishing property of our universal tile set is that it simulates nondeterministic choice in a "fair" way. An inherent feature of the Tile Assembly Model is the fact there are often multiple (say k) tiles that can go into any one position in an assembly sequence, and one of these k is nondeterministically chosen. One way to simulate this feature is to nondeterministically choose which of k supertiles should grow in the analogous (simulated) position. However, due to the size blowup in supertiles caused by encoding an arbitrary-sized simulated tile set into a fixed-sized universal tile set, it seems that we need to simulate one nondeterministic choice by using a sequence of nondeterministic choices within the supertile. Interpreting the nondeterministic choice to be made according to uniform random selection, if the selection by the simulating tile set is implemented in a naïve way, this can lead to unfair selection: when selecting 1 supertile out of k, some supertiles are selected with extremely low probability. To get around this problem, our system uses a random number selector that chooses a random tile with probability $\Theta(1/k)$ and so we claim that we are simulating nondeterminism in a "fair" way.

Thirdly, the Tile Assembly Model has certain geometric constraints that are not seen in cellular automata, and this adds some difficulty to our construction. Existing techniques for constructing intrinsically universal cellular automata are not directly applicable to tile assembly. For example, when a tile is placed at a position, that position can not be reused for further "computation" and this presents substantial difficulties when trying to fit the various components of our construction into a supertile. Each supertile encodes the entire simulated tile set and has the functionality to propagate this information to other (yet to be formed) supertiles. Not only this, each supertile must decide which tile placement to simulate, whilst making (fair) nondeterministic choices if necessary. Finally, each supertile should correctly propagate (output) sides that are consistent with the chosen supertile. We give a number of figures to illustrate how these goals were met within the geometric constraints of the model.

Our main result presented in this paper is, in some sense, a continuation of some previous results in self-assembly. For instance, Soloveichik and Winfree [23] exhibit a beautiful connection between the Kolmogorov complexity of a finite shape X and the minimum number of tiles types needed to assemble X. It turns out that their construction can be made to be "universal" in the following sense: there exists a tile set T, such that for every "temperature 1" tile assembly system that produces a finite shape whose underlying binding graph is a spanning tree, T simulates the given temperature 1 tile system with a corresponding blow-up in the scale. Note that this method restricts the simulated tile system to be temperature 1, i.e., a non-cooperative tile assembly system, which are conjectured [6] to produce "simple" shapes and patterns in the sense of Presburger arithmetic [17].

A similar result, recently discovered by Demaine, Demaine, Fekete, Ishaque, Rafalin, Schweller, and Souvaine [4], established the existence of a general-purpose "staged-assembly" system that is capable of simulating any temperature 1 tile assembly system that produces a "fully connected" finite shape. Note that, in this construction, the scaling factor is proportional to $O(\log |T|)$, where T is the simulated tile set. This construction has the desirable property that the set of tile types belonging to the simulator is general purpose (i.e., the size of the simulator tile set is independent of the to-be-simulated tile set) and all of the information needed to carry out the simulation is, in some sense, encoded in a sequence of laboratory steps. An open question in [4] is whether or not their construction can be augmented to handle temperature 2 tile assembly systems.

Our construction is general enough to be able to simulate powerful and interesting tile sets, yet sufficiently simple so that it actually belongs to the class of tile assembly systems that it can simulate, a class we term *locally consistent*. Systems in this class have the properties that each tile binds with exactly strength 2, and there are no glue mismatches in any producible assembly. This captures a wide class of tile assembly systems, including counters, square-builders and other shape-building tile assembly systems, and the tile assembly systems described in [1, 12, 19, 23]. Modulo re-scaling, our universal tile set can be said to display the characteristics of the entire collection of tile sets in its class. Our construction is a *direct simulation* in that the technique does not involve the simulation of intermediate models (such as circuits or Turing machines), which have been used in intrinsically universal cellular automata constructions [16].

One of the nice properties of intrinsic universality [16] is that it provides a clear definition that facilitates proofs that a given tile set is not universal. We leave as an open problem the intrinsic universality status of the Tile Assembly Model in its full generality.

Lafitte and Weiss [9–11] have also studied universality in the related model of Wang tiling [25]. Some of their definitions, particularly in [10], are similar to our definitions of simulation and universality, and also to those of Ollinger [16]. However, Wang tiling is not a model of self-assembly, as it is concerned with the ability of finite tile sets to tile the whole plane (with no mismatches), without regard to the *process* by which these tiles are placed. What is important is simply the existence of some valid tiling. In the TAM, which takes the order in which tiles are placed, one by one, into account, it must be shown that not only is there a sequence by which tiles could be individually and stably added to form the output assembly, but that *every* possible such sequence leads to the desired output. Furthermore, in the TAM a tile addition can be valid even if it causes mismatches as long as it is stable.

Most attempts to adapt the constructions of Wang tiling studies (such as those in [9–11]) to self-assembly result in a tile assembly system in which many junk assemblies are formed due to incorrect nondeterministic choices being made that arrest any further growth and/or result in assemblies which are inconsistent with the desired output assembly. We therefore require novel techniques to ensure that no nondeterminism is introduced, other than that already present in the tile system being simulated, and that the only produced assemblies are those that represent the intended result or valid partial progress toward it.

2. Intrinsic Universality in Self-Assembly

In this section, we define our notion of intrinsic universality of tile assembly systems. It is inspired by, but distinct from, similar notions for cellular automata [16]. Where appropriate, we identify where some part of our definition differs from the "corresponding" parts in [16], typically due to a fundamental difference between the abstract Tile Assembly Model and cellular automata models.

Intuitively, a tile set U is universal for a class $\mathfrak{C}$ of tile assembly systems if U can "simulate" any tile assembly system in $\mathfrak{C}$, where we use an appropriate seed assembly to give a tile assembly system $\mathcal{U}$. U is intrinsically universal if the simulation of $\mathcal{T}$ by $\mathcal{U}$ can be done according to a simple "block substitution scheme" where equal-size square blocks of tiles in assemblies produced by $\mathcal{U}$ represent tiles in assemblies produced by $\mathcal{T}$. Furthermore, since we wish to simulate the entire process of self-assembly, and not only the final result, it is critical that the simulation be such that the "local transition rules" involving intermediate producible (and nonterminal) assemblies of $\mathcal{T}$ be faithfully represented in the simulation.

In the subsequent definitions, given two partial functions f, g, we write $f(x) = g(x)$ if f and g are both defined and equal on x, or if f and g are both undefined on x. Let $c, c' \in \mathbb{N}$, let $[c : c']$ denote the set $\{c, c+1, \ldots, c'-1\}$, and let $[c]$ denote the set $[0 : c] = \{0, 1, \ldots, c-1\}$, so that $[c]^2$ forms a $c \times c$ square with the origin as the lower-left corner.

The natural analog of a configuration of a cellular automaton is an assembly of a tile assembly system. However, unlike cellular automata in which every cell has a well-defined state, in tile assembly, there is a fundamental difference between a point being empty space and being occupied by a tile. Therefore we keep the convention of representing an assembly as a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$ (for some tile set T), rather than treating empty space as just another type of tile.

Let $\mathcal{T} = (T, \sigma_\mathcal{T}, \tau)$ and $\mathcal{S} = (S, \sigma_\mathcal{S}, \tau)$ be tile assembly systems. For simplicity, assume that $\sigma_\mathcal{T}(0,0)$ is defined, and $\sigma_\mathcal{T}$ is undefined on $\mathbb{Z}^2 - \{(0,0)\}$ (i.e., $\mathcal{T}$ is singly-seeded with the seed tile placed at the origin). We will use this assumption of a single seed throughout the paper, but it is not strictly necessary and is only used for simplicity of discussion. Define a *representation function* to be a partial function of the form $r : ([c]^2 \dashrightarrow S) \dashrightarrow T$. That is, r takes a pattern $p : [c]^2 \dashrightarrow S$ of tile types from S painted onto a $c \times c$ square (with locations at which p is undefined representing empty space), and (if r is defined for input p) gives a single tile type from T. Intuitively, r tells us how to interpret $c \times c$ blocks within assemblies of $\mathcal{S}$ as single tiles of T. We write REPR for the set of all representation functions.

We say $\mathcal{S}$ *(intrinsically) simulates* $\mathcal{T}$ *with resolution loss* c if there exists a representation function $r : ([c]^2 \dashrightarrow S) \dashrightarrow T$ such that the following conditions hold.

(1) $\mathrm{dom}\, \sigma_\mathcal{S} \subseteq [c]^2$ and $r(\sigma_\mathcal{S}) = \sigma_\mathcal{T}(0,0)$, i.e., the seed assembly of $\mathcal{S}$ represents the seed of $\mathcal{T}$.

(2) For every producible assembly $\alpha_\mathcal{T} \in \mathcal{A}[\mathcal{T}]$ of $\mathcal{T}$, there is a producible assembly $\alpha_\mathcal{S} \in \mathcal{A}[\mathcal{S}]$ of $\mathcal{S}$ such that, for every $x, y \in \mathbb{Z}$,

$$r\left((\alpha_\mathcal{S} \upharpoonright ([cx : c(x+1)] \times [cy : c(y+1)])) + (-cx, -cy)\right) = \alpha_\mathcal{T}(x, y).$$

That is, the $c \times c$ block at (relative) position (x, y) (relative to the other $c \times c$ blocks; the absolute position is (cx, cy)) of assembly $\alpha_\mathcal{S}$ represents the tile type at (absolute) position (x, y) of assembly $\alpha_\mathcal{T}$. In this case, write $r^*(\alpha_\mathcal{S}) = \alpha_\mathcal{T}$; i.e., r induces a function $r^* : \mathcal{A}[\mathcal{S}] \to \mathcal{A}[\mathcal{T}]$.

(3) For all $\alpha_\mathcal{T}, \alpha'_\mathcal{T} \in \mathcal{A}[\mathcal{T}]$, it holds that $\alpha_\mathcal{T} \to_\mathcal{T} \alpha'_\mathcal{T}$ if and only if there exist $\alpha_\mathcal{S}, \alpha'_\mathcal{S} \in \mathcal{A}[\mathcal{S}]$ such that $r^*(\alpha_\mathcal{S}) = \alpha_\mathcal{T}$, $r^*(\alpha'_\mathcal{S}) = \alpha'_\mathcal{T}$ (in the sense of condition (2)), and $\alpha_\mathcal{S} \to_\mathcal{S} \alpha'_\mathcal{S}$. That is, every valid assembly sequence of $\mathcal{T}$ can be "mimicked" by $\mathcal{S}$, but no other assembly sequences can be so mimicked, so that the meaning of the relation $\to$ is preserved by r^*.

Let $\mathfrak{C}$ be a class of singly-seeded tile assembly systems, and let U be a tile set (with tile assembly systems having tile set U not necessarily elements of $\mathfrak{C}$). Note that every element of $\mathfrak{C}$, REPR, and $\mathsf{FIN}(U)$ is a finite object, hence can be represented in a suitable format for computation in some formal system such as Turing machines. We say U is *(intrinsically) universal* for $\mathfrak{C}$ if there are computable functions $R : \mathfrak{C} \longrightarrow \mathsf{REPR}$ and $A : \mathfrak{C} \longrightarrow \mathsf{FIN}(U)$ such that, for each $\mathcal{T} = (T, \sigma_\mathcal{T}, \tau) \in \mathfrak{C}$, there is a constant $c \in \mathbb{N}$ such that, letting $r = R(\mathcal{T})$, $\sigma = A(\mathcal{T})$, and $\mathcal{U}_\mathcal{T} = (U, \sigma, \tau)$, $\mathcal{U}_\mathcal{T}$ simulates $\mathcal{T}$ with resolution loss c and representation function r. That is, $R(\mathcal{T})$ outputs a representation function that interprets assemblies of $\mathcal{U}_\mathcal{T}$ as assemblies of $\mathcal{T}$, and $A(\mathcal{T})$ outputs the seed assembly used to program tiles from U to represent the seed tile of $\mathcal{T}$.

3. An Intrinsically Universal Tile Set

In this section, we exhibit an intrinsically universal tile set for any "nice" tile assembly system. Before proceeding, we must first define the notion of a "nice" tile assembly system. Let $\mathcal{T} = (T, \sigma, 2)$ be a tile assembly system, and $\vec{\alpha}$ be an assembly sequence in $\mathcal{T}$ whose result is denoted as α. We say that $\mathcal{T}$ is *locally consistent* if the following conditions hold.

(1) For all $\vec{m} \in \mathrm{dom}\, \alpha - \mathrm{dom}\, \sigma$, $\sum_{\vec{u} \in \mathrm{IN}^{\vec{\alpha}}(\vec{m})} \mathrm{str}_{\alpha(\vec{m})}(\vec{u}) = 2$, where $\mathrm{IN}^{\vec{\alpha}}(\vec{m})$ is the set of sides on which the tile that $\vec{\alpha}$ places at location $\vec{m}$ initially binds. That is, every tile initially binds to the assembly with exactly bond strength equal to 2 (either a single strength 2 bond or two strength 1 bonds).

(2) For all producible assemblies $\alpha \in \mathcal{A}[\mathcal{T}]$, $\vec{u} \in U_2$, and $\vec{m} \in \text{dom } \alpha$, if $\alpha(\vec{m} + \vec{u})$ is defined, then the following condition holds:

$$\text{str}_{\alpha(\vec{m})}(\vec{u}) > 0 \Rightarrow \text{label}_{\alpha(\vec{m})}(\vec{u}) = \text{label}_{\alpha(\vec{m}+\vec{u})}(-\vec{u}) \text{ and } \text{str}_{\alpha(\vec{m})}(\vec{u}) = \text{str}_{\alpha(\vec{m}+\vec{u})}(-\vec{u}).$$

While condition (1) of the above definition is reminiscent of the first condition of local determinism [23], the second condition says that there are no (positive strength) label mismatches between abutting tiles. However, we must emphasize that a locally consistent tile assembly system need not be directed, and moreover, even a locally deterministic tile assembly system need not be locally consistent because of the lack of any kind of "determinism restriction" in the latter definition. Our main result is the following.

Theorem 3.1 (Main theorem)**.** *Let $\mathfrak{C}$ be the set of all locally consistent tile assembly systems. There exists a finite tile set U that is intrinsically universal for $\mathfrak{C}$.*

In the remainder of this section, we prove Theorem 3.1, that is, we show that for every locally consistent tile assembly system $\mathcal{T} = (T, \sigma, 2)$, there exists a seed assembly $\sigma_{\mathcal{T}}$, such that the tile assembly system $\mathcal{U}_{\mathcal{T}} = (U, \sigma_{\mathcal{T}}, 2)$ simulates $\mathcal{T}$ with a resolution loss $c \in \mathbb{N}$ that depends only on the glue complexity of $\mathcal{T}$. Instead of giving an explicit (and tedious) definition of the tile types in U, we implicitly define U by describing how $\mathcal{U}_{\mathcal{T}}$ simulates $\mathcal{T}$.

3.1. High-Level Overview

Intuitively, $\mathcal{U}$ simulates $\mathcal{T}$ by growing "supertiles" that correspond to tile types in T. In other words, every supertile is a $c \times c$ block of tiles that is mapped to a tile type $t \in T$. To do this, each supertile that assembles in $\mathcal{U}_{\mathcal{T}}$ contains the full specification of T as a lookup table (a long row of tiles that encodes all of the information in the set of tile types T), analogous to the genome of an organism being fully replicated in each cell of that organism, no matter how specialized the function of the cell. This lookup table is carefully propagated through each supertile in $\mathcal{U}_{\mathcal{T}}$ via a series of "rotation" and "copy" operations – both of which are well-known self-assembly primitives.

In the table, we represent each (glue,direction) pair as a binary string, and represent the tile set as a table mapping 1-2 input glue(s) to 0-3 output glue(s). Since each tile type of $\mathcal{T}$ may not have well-defined input sides, when two supertiles representing tiles of $\mathcal{T}$ must potentially cooperate to place a new supertile within a block adjacent to both of them, it is imperative that each grows into the block in such a way as to remain unobtrusive to the other supertile. This is done with a "probe" that grows toward the center of the block, as shown in Figure 1. At the moment the probes meet in the middle, they "find out" in what direction the other input supertile lies, and at that point decide in which direction to grow the rest of the forming supertile. so as to avoid the tiles that were already placed as part of the probes. We do not know how to deal with three probes at once, which is the reason both parts of the definition of locally consistent, which imply that only two input probes will ever be present at one time. The next step is to bring the values of two input glues together before doing a lookup on the table, because they are both needed to simulate cooperation. The table must be read and copied at the same time, otherwise the planarity of the tiles would hide the table as it is read and it could not be propagated to the output supertiles. Many choices made in the construction, such as the relative positioning of glues/table, or the counter-clockwise order of assembly, are choices that simply were convenient and seemed to work, but are not necessarily required.

3.2. Construction of the Lookup Table

In order to simulate the behavior of $\mathcal{T}$ with $\mathcal{U}_{\mathcal{T}}$, we must first encode the definition of T using tiles from U. We will do this by constructing a "glue lookup table," denoted as $\mathbf{T}_{\mathcal{T}}$, and is essentially the self-assembly version of a kind of hash table. Informally, $\mathbf{T}_{\mathcal{T}}$ is a (very) long string (of tiles from U) consisting of two copies of the definition of the tile set T separated by a small group of spacer symbols. The left copy of the lookup table is the *reverse* of the right copy. The lookup table maps all possible sets of input sides for each tile type $t \in T$ to the corresponding sets of output sides.

3.2.1. *Addresses.* The lookup table $\mathbf{T}_{\mathcal{T}}$ consists of a contiguous sequence of "addresses," which are formed from the definition of T. Namely, for each tile type $t \in T$, we create a unique binary key for each combination of sides of t whose glue strengths sum to exactly 2. Each of these combinations represents a set of sides which *could* potentially serve as the input sides for a tile of type t in a producible assembly in $\mathcal{T}$.

We say that a *pad* is an ordered triple (g, d, s) where g is a glue label in T, $d \in \{N, S, E, W\}$ is an edge direction, and $s \in \{0, 1, 2\}$ is an allowable glue strength. Note that a set of four pads – one for each direction d – fully specifies a tile type. We use $\mathrm{Pad}(t, d)$ to denote the pad on side d of the tile type $t \in T$

Let $\mathrm{Bin}(p)$ be the binary encoding of a pad $p = (g, d, s)$, consisting of the concatenation of the following component binary strings:

(1) g (*glue specification*): Let G be the set of glue types from all edges with positive glue strengths in $T \cup \{g_{\mathrm{null}}\}$ (a.k.a., the null glue). Fix some ordering $g_{\mathrm{null}} \leq g_0 \leq g_1 \leq \cdots$ of the set G. The binary representation of g_i is the binary value of i padded with 0's to the left (as necessary) to ensure that the string is exactly $\lceil \log(|G|+1) \rceil$ bits.

(2) d (*direction*): If $d = N$ (E, S, or W), append 00 (01, 10, or 11, respectively).

(3) s (*strength*): If $s = 1$ (2) append 0 (1).

Note that $\lceil \log(|G|+1) \rceil + 2 + 1$ is the length of the binary string encoding an arbitrary pad p, and is a constant that depends only on T.

An *address* is a binary string that represents a set of pads which, themselves, can potentially serve as the input sides of some tile type $t \in T$. It can be composed of one of the two following binary strings:

(1) A prefix of zeros, $0^{\lceil \log(|G|+1) \rceil + 3}$, followed by $\mathrm{Bin}(p)$ for $p = (g, d, 2)$, or

(2) the concatenation of $\mathrm{Bin}(p_1)$ and $\mathrm{Bin}(p_2)$ for $p_1 = (g_1, d_1, 1)$ and $p_2 = (g_2, d_2, 1)$. The ordering of $\mathrm{Bin}(p_1)$ and $\mathrm{Bin}(p_2)$ in an address must be consistent with the following orderings: EN, SE, WS, NW, NS, EW.

Note that it is possible for more than one tile type $t \in T$ to share a set of input pads and therefore an address.

3.2.2. *Encoding of* T. We will now construct the string $w_{\mathcal{T}}$, which will represent the definition of T. Intuitively, $w_{\mathcal{T}}$ will be composed of a series of "entries." Each entry is associated to exactly one address of a tile type $t \in T$ and specifies the pads for the output sides of t. In this way, once the input sides for a supertile have formed, the corresponding pads can be used to form an address specifying (a set of) appropriate output pads. Note that since more than one tile type may share an address in a nondeterministic tile assembly system, more than one tile type may share a single entry.

We define an *entry* to be a string beginning with '#' followed by zero or more "sub-entries", each corresponding to a different tile type, separated by semicolons. Let A be the set of all binary strings representing every address created for each $t \in T$. The string $w_{\mathcal{T}}$ will consist of $1 + \max A$ entries for addresses 0 to $\max A$. The i^{th} entry, denoted as e_i, corresponds to the i^{th} address, which may or may not be in A (if it is not, then e_i is empty).

We say that a *sub-entry* consists of a string specifying the pads for the output sides of a tile type $t \in T$. Let e_i be the entry containing a given sub-entry (note that i is the address of e_i), and $T_i \subseteq T$ be the set of tile types addressable by i (i.e., the set of tile types for which i is a valid address). The entry e_i will be comprised of exactly $|T_i|$ sub-entries. For $0 \leq k < j$, the k^{th} sub-entry in e_i, where $t_k \in T_i$ is the k^{th} element of T_i (relative to some fixed ordering), is the string $\mathrm{OUT}(N), \mathrm{OUT}(E), \mathrm{OUT}(S), \mathrm{OUT}(W)$ (the commas in the previous string are literal) with $\mathrm{OUT}(d) = \mathrm{Bin}(\mathrm{Pad}(t_k, d))^R$ if the glue for $\mathrm{Pad}(t_k, d)$ is not g_{null} and d is not a component of the address i, otherwise $\mathrm{OUT}(d) = \lambda$. Intuitively, a sub-entry is a comma-separated list of the (reversed) binary representations of the pads for an addressed tile type, but including only pads whose glues are not g_{null} and whose directions are not a part of the address (and therefore input sides). We will now use the string $w_{\mathcal{T}}$ to construct the lookup table $\mathbf{T}_{\mathcal{T}}$.

3.2.3. *Full specification of* $\mathbf{T}_{\mathcal{T}}$. We now give the full specification for the lookup table $\mathbf{T}_{\mathcal{T}}$. First, define the following strings: $w_0 =$ '$>$', $w_1 =$ '$< \%\% >$', $w_2 =$ '$<$'. Now let $\mathbf{T}_{\mathcal{T}}$ be as follows: $\mathbf{T}_{\mathcal{T}} = \mathrm{sb}(w_0 \circ w_{\mathcal{T}} \circ w_1 \circ (w_{\mathcal{T}})^R \circ w_2)$, where, for strings x and y, $x \circ y$ is the concatenation of x and y, and $\mathrm{sb} : \Sigma^* \to \Sigma^*$ is defined to "splice blanks" into its input: between every pair of adjacent symbols in the string x, a single '␣' (blank) symbol is inserted to create $\mathrm{sb}(x)$. This splicing of blanks is required to be able to read from the table without "locking it from view", when reading the table for operations that require growing a column of tiles in towards the table (as opposed to away from it), a blank column is used, and for growing a column away from the table, a symbol column is used so that the symbol can be propagated to the top of the column for later copying.

3.2.4. *The Lookup Procedure.* In our construction, when a supertile t^* that is simulating a tile type $t \in T$ forms, we must overcome the following problem: once we combine the input pads (given as the output pads of the supertiles to which t^* attaches), how do we use $\mathbf{T}_{\mathcal{T}}$ to lookup the output pads for t^*? In what follows, we briefly describe how we achieve this. In other words, we show how an address, a string of random bits, and a copy of $\mathbf{T}_{\mathcal{T}}$ are used to compute the pad values for the non-input sides of a supertile. A detailed figure and example of this procedure can be found in the technical appendix.

For ease of discussion and without loss of generality, we assume that the row of tiles encoding $\mathbf{T}_{\mathcal{T}}$ (assembled West to East) and the column of tiles encoding an address and a random string of bits (assembled North to South at the West end of $\mathbf{T}_{\mathcal{T}}$) are fully assembled, forming an 'L' shape with no tiles in the area between them. For other orientations of the table and address the logical behavior is identical, simply rotated.

Intuitively, the assembly of the lookup procedure assembles column wise in a zig-zag fashion from left to the right. In the "first phase," a counter initialized to 0 is incremented in each column where the value of the tile in the representation of $\mathbf{T}_{\mathcal{T}}$ is a ';', thus counting up at each entry contained in $\mathbf{T}_{\mathcal{T}}$. Once that number matches the value of the given address (which, along with the random bits is copied through this procedure), the entry e

corresponding to that address has been reached and a new counter begins which counts the number of sub-entries n in that entry. Note that for directed tile systems, $n \leq 1$. Once the end of that entry is encountered, yet another counter, initialized to 0, begins and increments on each remaining entry until the end of the first copy of $w_{\mathcal{T}}$ is reached (the number n is propagated to the right). This counts the number of entries, denoted as m, between e and the end of the lookup table. The "second phase" is used to perform, in some sense, an operation equivalent to calculating $p = b \mod n$, where b is the binary value of the string of random bits required for the lookup procedure (this is how we simulate nondeterministic assemblies). This selects the index of the sub-entry in e which will be used, completing the random selection of one of the possibly many tile types contained in entry e.

In the current version of our construction, we merely use a random number selection procedure reminiscent of the more involved (but more uniform) random selection procedures discussed in [5]. Although it is possible to incorporate these more advanced techniques into our construction (and thus achieve a higher degree of uniformity in the simulation of randomized tile systems), we choose not to do so for the sake of simplicity.

Next, a reverse counter, a.k.a., a subtractor, counts down at each entry from m to 0, and by the way we constructed $\mathbf{T}_{\mathcal{T}}$, this final counter obtains the value 0 at the entry e (in the reverse of $w_{\mathcal{T}}$). Now, another subtractor counts from p to 0 to locate the correct sub-entry that was selected randomly. Finally, each pad in the sub-entry is rotated "up and to the right," and the group of pads is propagated through the remainder of the lookup table, thus ending with the values of the non-input pads represented in the rightmost column.

3.3. Supertile design

A supertile s is a subassembly in $\mathcal{U}_{\mathcal{T}}$ consisting of a $c \times c$ block of tiles from T, where c depends on the glue complexity of T. Each s can be mapped to a unique tile type $t \in T$. In our construction there are two logical supertile designs. The first, denoted *type-0*, simulates tile additions in $\mathcal{T}$ in which there are 2 input sides, each with glue strength $= 1$. The second, denoted *type-1*, simulates the addition of tiles via a single strength 2 bond.

While there are several differences in the designs of *type-0* and *type-1* supertiles, one commonality is how their edges are defined. Namely each input or output edge of any supertile is defined by the same sequence of variable values. Since the edges for each direction are rotations of each other, we will discuss only the layout of the south side of a supertile. From left to right, the tiles along the south edge of a supertile will represent a string formed by the concatenation (in order) of the strings: $\mathbf{T}_{\mathcal{T}}$, $\text{Bin}(\text{Pad}(t, S))$, $0^{c'}$, $\text{Bin}(\text{Pad}(t, S))$, and $\mathbf{T}_{\mathcal{T}}$. Note that c' is a constant that depends on the glue complexity of T.

3.3.1. *Type-0 Supertiles (i.e., simulating tiles that attach via two single-strength bonds).* When a tile binds to an assembly in $\mathcal{T}$ with two input sides whose glues are each single strength, there are $\binom{4}{2} = 6$ possible combinations of directions for those input sides: north and east (NE), north and south (NS), north and west (NW), east and south (ES), east and west (EW), and south and west (SW). These combinations can be divided into two categories, those in which the sides are opposite each other (NS and EW), and those in which the sides are adjacent to each other (NE, NW, ES, and SW).

Opposite Input Sides: Supertiles which represent tile additions with two opposite input sides, NS and EW, are logically identical to rotations of each other, so here we will only describe the details of a supertile with NS input sides. Figure 1 shows a detailed image

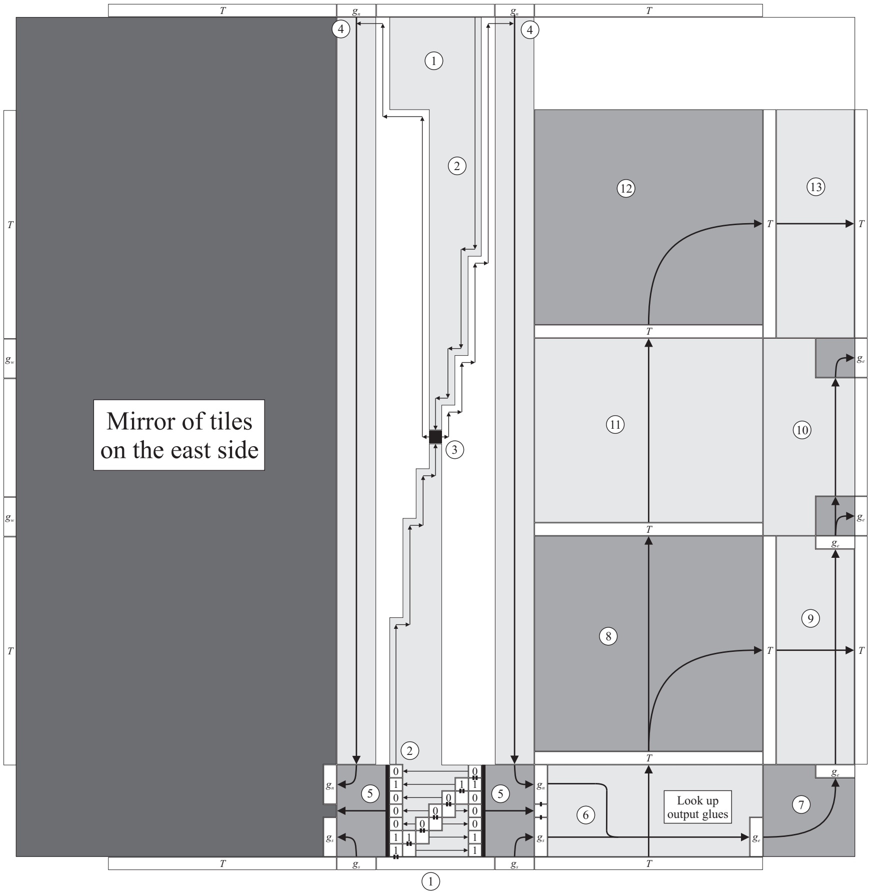


Figure 1: NS supertile

depicting the formation of an NS supertile, with arrows giving the direction of growth for each portion and numbers specifying the order of growth. For ease of discussion and without loss of generality, we assume that the rows of tiles which form the input sides of a supertile have fully formed before any other part of the supertile assembles. The first portions to assemble are the center blocks to the interior of each input side, labeled 1. This subassembly forms a square in which a series of nondeterministic selections of tile types is used to generate a random sequence of bits. These bits are propagated to the left and right sides of the block, to ensure that each side uses the same random bits for the randomized selection after the sides have been "sealed off" from each other by "probes" described next. Once that block has completed, a log-width binary subtractor, which is half the width of the block, assembles. The subtractors from the north and south count down from a specified value (that depends on $\mathcal{T}$ and is encoded into the seed supertile) to 0, and shrink in width until they terminate at positions adjacent to the center square of the block. These subtractors are "probes" that grow to the center where the direction of the input sides (the *type*) is detected. It is at this point that the central (black in the figure) tile can attach. It is this tile which determines the *type* of the supertile (NS in this case) because it is unique to the combination of directions from which the inputs came. At this

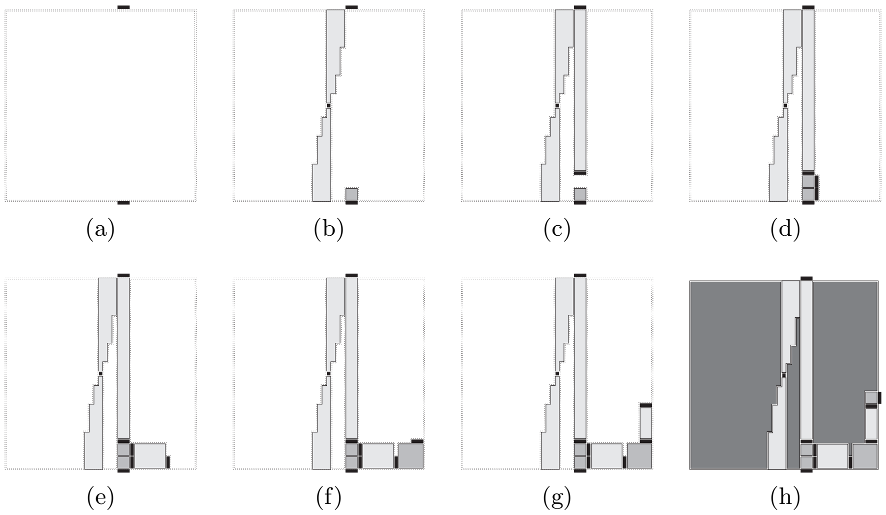


Figure 2: Intuitive depiction of (a portion of) the self-assembly of a type-0 supertile. Note that the lookup procedure is performed in (d) and (e).

point, symmetry is broken and two paths of tiles assemble from the center back towards the north side. They in turn initiate the growth of subassemblies which propagate the value of the north input pad down towards the South of the supertile. Once that growth nears the southern side, the two input pads are rotated and brought together, with this combination of input pads forming an *address* in the lookup table. In the manner described previously, this address along with the random bits generated within block 1 (which are also passed through block 5) is used to form the subassembly of block 6 whose southern row contains a representation of $\mathbf{T}_{\mathcal{T}}$ and results in the correct output pads being represented in the final column of that block. Note that Figure 1 only shows the details of the east side of the block since the West side is an identical but rotated version. Finally, subassemblies 7 through 13 form which rotate and pass the necessary information to the locations where it must be correctly deposited to form the output sides of the supertile. Every side of a supertile that is not an input side receives an output pad, even if it is for the null glue (in which case it does not initiate the growth of the input side of a possible adjacent supertile).

References

1. Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang, *Running time and program size for self-assembled squares*, STOC '01: Proceedings of the thirty-third annual ACM Symposium on Theory of Computing (New York, NY, USA), ACM, 2001, pp. 740–748.
2. J. Albert and K. Čulik II, *A simple universal cellular automaton and its one-way and totalistic version*, Complex Systems **1** (1987), no. 1, 1–16.
3. Ho-Lin Chen and Ashish Goel, *Error free self-assembly with error prone tiles*, Proceedings of the 10th International Meeting on DNA Based Computers, 2004.
4. Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine, *Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues*, Natural Computing **7** (2008), no. 3, 347–370.

5. David Doty, Jack H. Lutz, Matthew J. Patitz, Scott M. Summers, and Damien Woods, *Random number selection in self-assembly*, Proceedings of The Eighth International Conference on Unconventional Computation (Porta Delgada (Azores), Portugal, September 7-11, 2009), 2009.
6. David Doty, Matthew J. Patitz, and Scott M. Summers, *Limitations of self-assembly at temperature 1*, Proceedings of The Fifteenth International Meeting on DNA Computing and Molecular Programming (Fayetteville, Arkansas, USA, June 8-11, 2009), 2009, to appear.
7. B. Durand and Zs. Róka, *The game of life: universality revisited*, Tech. Report 98-01, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, January 1998.
8. Kenichi Fujibayashi, David Yu Zhang, Erik Winfree, and Satoshi Murata, *Error suppression mechanisms for dna tile self-assembly and their simulation*, Natural Computing: an international journal **8** (2009), no. 3, 589–612.
9. Grégory Lafitte and Michael Weiss, *Universal tilings*, STACS (Wolfgang Thomas and Pascal Weil, eds.), Lecture Notes in Computer Science, vol. 4393, Springer, 2007, pp. 367–380.
10. ______, *Simulations between tilings*, Tech. report, University of Athens, 2008.
11. ______, *An almost totally universal tile set*, TAMC (Jianer Chen and S. Barry Cooper, eds.), Lecture Notes in Computer Science, vol. 5532, Springer, 2009, pp. 271–280.
12. James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers, *Computability and complexity in self-assembly*, Proceedings of The Fourth Conference on Computability in Europe (Athens, Greece, June 15-20, 2008), 2008.
13. Urmi Majumder, Thomas H LaBean, and John H Reif, *Activatable tiles for compact error-resilient directional assembly*, 13th International Meeting on DNA Computing (DNA 13), Memphis, Tennessee, June 4-8, 2007., 2007.
14. Maurice Margenstern, *Cellular automata in hyperbolic spaces, vol. 1: Theory*, Old City Publishing, Philadelphia, 2007.
15. Nicolas Ollinger, *The intrinsic universality problem of one-dimensional cellular automata*, 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS) (H. Alt and M. Habib, eds.), LNCS, vol. 2607, Springer, 2003, pp. 632–641.
16. ______, *Intrinsically universal cellular automata*, Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6-7th December 2008 (T. Neary, D. Woods, A.K. Seda, and N. Murphy, eds.), EPTCS, vol. 1, 2009, arXiv:0906.3213v1 [cs.CC], pp. 199–204.
17. Mojżesz Presburger, *Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen*, welchem die Addition als einzige Operation hervortritt. Compte Rendus du I. Congrks des Mathematiciens des pays Slavs, Warsaw, 1930, pp. 92–101.
18. Paul W. K. Rothemund, *Theory and experiments in algorithmic self-assembly*, Ph.D. thesis, University of Southern California, December 2001.
19. Paul W. K. Rothemund and Erik Winfree, *The program-size complexity of self-assembled squares (extended abstract)*, STOC '00: Proceedings of the thirty-second annual ACM Symposium on Theory of Computing (New York, NY, USA), ACM, 2000, pp. 459–468.
20. Paul W.K. Rothemund, Nick Papadakis, and Erik Winfree, *Algorithmic self-assembly of DNA Sierpinski triangles*, PLoS Biology **2** (2004), no. 12, 2041–2053.
21. Nadrian C. Seeman, *Nucleic-acid junctions and lattices*, Journal of Theoretical Biology **99** (1982), 237–247.
22. David Soloveichik and Erik Winfree, *Complexity of compact proofreading for self-assembled patterns*, The eleventh International Meeting on DNA Computing, 2005.
23. ______, *Complexity of self-assembled shapes*, SIAM Journal on Computing **36** (2007), no. 6, 1544–1569.
24. Thomas LaBean Urmi Majumder, Sudheer Sahu and John H. Reif, *Design and simulation of self-repairing DNA lattices*, DNA Computing: DNA12, Lecture Notes in Computer Science, vol. 4287, Springer-Verlag, 2006.
25. Hao Wang, *Proving theorems by pattern recognition – II*, The Bell System Technical Journal **XL** (1961), no. 1, 1–41.
26. Erik Winfree, *Algorithmic self-assembly of DNA*, Ph.D. thesis, California Institute of Technology, June 1998.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 287-298
www.stacs-conf.org

SPONSORED SEARCH, MARKET EQUILIBRIA, AND THE HUNGARIAN METHOD

PAUL DÜTTING [1] AND MONIKA HENZINGER [1,2] AND INGMAR WEBER [1,3]

[1] Ecole Polytechnique Fédérale de Lausanne, Switzerland
E-mail address: {paul.duetting,monika.henzinger,ingmar.weber}@epfl.ch

[2] University of Vienna, Austria
E-mail address: monika.henzinger@univie.ac.at

[3] Yahoo! Research Barcelona, Spain
E-mail address: ingmar@yahoo-inc.com

Abstract. Two-sided matching markets play a prominent role in economic theory. A prime example of such a market is the sponsored search market where n advertisers compete for the assignment of one of k sponsored search results, also known as "slots", for certain keywords they are interested in. Here, as in other markets of that kind, market equilibria correspond to stable matchings. In this paper, we show how to modify Kuhn's Hungarian Method (Kuhn, 1955) so that it finds an optimal stable matching between advertisers and advertising slots in settings with generalized linear utilities, per-bidder-item reserve prices, and per-bidder-item maximum prices. The only algorithm for this problem presented so far (Aggarwal et al., 2009) requires the market to be in "general position". We do not make this assumption.

1. Introduction

Two-sided matching markets play a prominent role in economic theory. A prime example of such a market is the *sponsored search market* [14] where n advertisers (or bidders) compete for the assignment of one of k sponsored search results, also known as "slots", for certain keywords (or items) they are interested in. Here, as in other markets of that kind, *market equilibria* correspond to *stable matchings*. A stable matching that is preferred by all bidders over all other stable matchings is *bidder optimal*. Mechanisms that compute bidder optimal matchings typically provide the bidders with the incentive to reveal their *true* preferences, i.e., they are *truthful*.

In the most basic model of a two-sided matching market, known as the *stable marriage problem* [9], each bidder has a strict preference ordering over the items and each item has a strict preference ordering over the bidders. In a more general model, see e.g. [16], each bidder has a linear utility function for each item that depends on the price of the item and

1998 ACM Subject Classification: F.2.2 (Nonnumerical Algorithms and Problems).
Key words and phrases: stable matching, envy-free allocation, general auction mechanism, general position.
This work was conducted as part of a EURYI scheme award (see http://www.esf.org/euryi/).

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2463

every item can have a *reserve price*, i.e., a price under which the item cannot be sold to *any* bidder. In the even stronger model that we study here every bidder-item pair can have a *reserve price*, i.e., a price under which the item cannot be sold to this *specific* bidder, and a *maximum price*, i.e., a price above which this bidder does not want to buy this *specific* item. We call this model the *sponsored search market.* An interesting property of this model is that it generalizes standard auction formats such as *VCG* [17, 4, 10] and *GSP* [7].

While the problem of finding a bidder optimal matching in the first two models has been largely solved in the 60s, 70s, and 80s [9, 16, 5, 15], the problem of finding a bidder optimal matching in the sponsored search market has been addressed only recently [2].

The main finding of [2] is that if the market is in "general position", then (a) there is a unique bidder optimal matching and (b) it can be found in $O(nk^3)$ steps by a truthful mechanism. For a market to be in "general position", however, any two reserve prices and/or maximum prices must be distinct. In practice, this will rarely be the case and so we typically have to deal with markets that are *not* in general position. The authors of [2] propose to bring such markets into "general position" using random perturbations and/or symbolic tie-breaking. The problem with this approach, however, is that there is no guarantee that a bidder optimal solution of the perturbed market leads to a bidder optimal solution of the original market. In fact, such a solution may not even exist (see Section 3). Additionally, a pertubation-based mechanism may not be truthful.

We improve upon the results of [2] as follows: First, in Section 3, we show how to modify the definition of stability so that a bidder optimal matching is guaranteed to exist for arbitrary markets. Then, in Section 5, 6, and 7, we show how to modify Kuhn's *Hungarian Method* [13, 8] so that it finds a bidder optimal matching in time $O(nk^3 \log(k))$. Afterwards, in Section 8, we show that with our notion of stability bidder optimality no longer implies truthfulness, unless further restrictions are imposed on the model. Finally, in Section 9, we show how to reduce more general linear utility functions to our setting.[1]

Independently of us Ashlagi et al. [3] also improved upon the results of [2] by (a) showing the existence of a unique feasible, envy free, and Pareto efficient solution for *position auctions with budgets* and by (b) providing a truthful mechanism that finds it. The notion of envy-freeness is equivalent to our notion of stability. Their model, however, is a special case of our model as it requires a common preference ordering over the items, it does not incorporate reserve prices, it does not allow the maximum prices to depend on the bidder *and* the item, and it requires the maximum prices to be distinct.

Recently, Kempe et al. [12] presented an efficient algorithm that finds the minimum envy-free prices (if they exist) for a *given* matching.

To summarize our main contributions are: (1) We show how to modify the Hungarian Method so that it finds a bidder optimal solution for *arbitrary* markets, including markets that are *not* in "general position". (2) We show how different definitions of stability affect the existence of a bidder optimal solution. (3) We show how to reduce more general linear utility functions to the setting that we study in this paper with no loss in performance.

2. Problem Statement

We are given a set I of n bidders and a set J of k items. We use letter i to denote a bidder and letter j to denote an item. For each bidder i and item j we are given a *valuation*

[1] These utilities can be used to model that the *click probability* in the *pay-per-click model* has a bidder-dependent component c_i and an item-dependent component c_j. See [1, 7] for details.

$v_{i,j}$, a *reserve price* $r_{i,j}$, and a maximum price $m_{i,j}$. We assume that the set of items J contains a *dummy item* j_0 for which all bidders have a valuation of zero, a reserve price of zero, and a maximum price of ∞.[2]

We want to compute a matching $\mu \subseteq I \times J$ and per-item prices $p = (p_1, \dots, p_k)$. We require that every bidder i appears in exactly one bidder-item pair $(i, j) \in \mu$ and that every non-dummy item $j \neq j_0$ appears in at most one such pair. We allow the dummy item j_0 to appear more than once. We call bidders (items) that are not matched to any non-dummy item (bidder) *unmatched.* We regard the dummy item as unmatched.

We define the *utility* u_i of bidder i to be $u_i = 0$ if bidder i is unmatched and $u_i = u_{i,j}(p_j)$ if bidder i is matched to item j at price p_j. We set $u_{i,j}(p_j) = v_{i,j} - p_j$ if $p_j < m_{i,j}$ and $u_{i,j}(p_j) = -\infty$ if $p_j \geq m_{i,j}$. We say that a matching μ with prices p is *feasible* if (1) $u_i \geq 0$ for all i, (2) $p_{j_0} = 0$ and $p_j \geq 0$ for all $j \neq j_0$, and (3) $r_{i,j} \leq p_j < m_{i,j}$ for all $(i, j) \in \mu$. We say that a feasible matching μ with prices p is *stable* if $u_i \geq u_{i,j}(p_j)$ for all $(i, j) \in I \times J$.[3] Finally, we say that a stable matching μ with prices p is *bidder optimal* if $u_i \geq u'_i$ for all i and stable matchings μ' with prices p'.

We say that an algorithm is *truthful* if for every bidder i with utility functions $u_{i,1}(\cdot), \dots, u_{i,k}(\cdot)$ and any two inputs $(u'_{i,j}(\cdot), r_{i,j}, m'_{i,j})$ and $(u''_{i,j}(\cdot), r_{i,j}, m''_{i,j})$ with $u'_{i,j}(\cdot) = u_{i,j}(\cdot)$ for i and all j and $u'_{k,j}(\cdot) = u''_{k,j}(\cdot)$ for $k \neq i$ and all j and matchings μ' with p' and μ'' with p'' we have that $u_{i,j'}(p'_{j'}) \geq u_{i,j''}(p''_{j''})$ where $(i, j) \in \mu$ and $(i, j'') \in \mu''$. This definition formalizes the notion that "lying does not pay off" as follows: Even if bidder i claims that his utility is $u''_{i,j}$ instead of $u_{i,j}$ he will not achieve a higher utility with the prices and the matching computed by the algorithm. Thus, the algorithm "encourages truthfulness".

3. Motivation

The definition of stability in [2], which we call *relaxed stability* to indicate that every stable solution is also relaxed stable (but not vice versa), requires that for every pair $(i, j) \in I \times J$ either (a) $u_i \geq v_{i,j} - \max(p_j, r_{i,j})$ or (b) $p_j \geq m_{i,j}$. The disadvantage of relaxed stability is that there can be situations where no bidder optimal solution exists if the market is *not* in "general position" (see [2] for a formal definition). Here are two canonical examples:

- *Example 1.* There are three bidders and two items. The valuations and reserve prices are as follows: $v_{1,1} = 1$, $v_{2,1} = 4$, $v_{2,2} = 4$, $v_{3,2} = 1$, $r_{1,1} = 0$, $r_{2,1} = r_{2,2} = 2$, and $r_{3,2} = 0$. While $\mu = \{(1, 1), (2, 2)\}$ with $p = (0, 2)$ is "best" for bidder 1, $\mu = \{(2, 1), (3, 2)\}$ with $p = (2, 0)$ is "best" for bidder 3.
- *Example 2.* There are two bidders and one item. The valuations and maximum prices are as follows: $v_{1,1} = 10$, $v_{2,1} = 10$, and $m_{1,1} = m_{2,1} = 5$. While $\mu = \{(1, 1)\}$ with $p_1 = 5$ is "best" for bidder 1, $\mu = \{(2, 1)\}$ with $p_1 = 5$ is "best" for bidder 2.

In the market of the first example no bidder optimal solution exists as long as there exists a bidder that has the same utility functions and reserve prices for two items and two other bidders that are only interested in one of the items. In the market of the second example no bidder optimal solution exists as long as both bidders have the same maximum price and a non-zero utility at the maximum price. Since these cases are quite general, we conjecture that they occur rather frequently in practice.

[2] Reserve utilities, or *outside options* o_i, can be incorporated by setting $v_{i,j_0} = o_i$ for all bidders i.

[3] Since we have $u_i \geq 0$ and $u_{i,j}(p_j) = -\infty$ if $p_j \geq m_{i,j}$, this definition is equivalent to requiring $u_i \geq v_{i,j} - p_j$ for all items j with $p_j < m_{i,j}$.

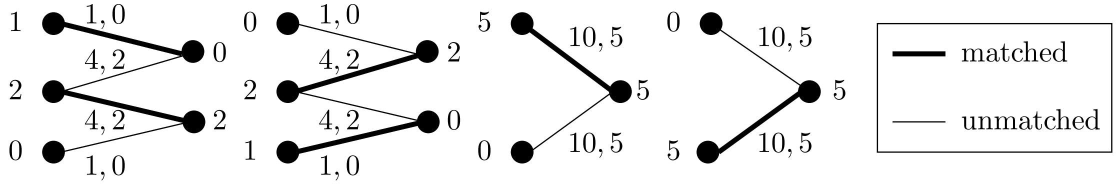


Figure 1: The left two graphs illustrate Example 1. The right two graphs illustrate Example 2. Bidders are on the left side, items on the right side of the bipartite graph. The numbers next to the bidder indicate her utility, the numbers next to the item indicate its price. The labels along the edge show valuations and reserve prices for the left two graphs and valuations and maximum prices for the right two graphs. With relaxed stability a bidder optimal matching does not exist.

With our notion of stability a bidder optimal solution is guaranteed to exist (e.g. $\mu = \{(2,1)\}$ with $p_1 = p_2 = 2$ in Example 1 and $\mu = \emptyset$ with $p_1 = 5$ in Example 2) for *all* kinds of markets, including markets that are *not* in general position.

4. Preliminaries

We define the *first choice graph* $G_p = (I \cup J, F_p)$ at prices p as follows: There is one node per bidder i, one node per item j, and an edge from i to j if and only if item j gives bidder i the highest utility possible, i.e., $u_{i,j}(p_j) \geq u_{i,j'}(p_{j'})$ for all j'. For $i \in I$ we define $F_p(i) = \{j : \exists\ (i,j) \in F_p\}$ and similarly $F_p(j) = \{i : \exists\ (i,j) \in F_p\}$. Analogously, for $T \subseteq I$ we define $F_p(T) = \cup_{i \in T} F_p(i)$ and for $S \subseteq J$ we define $F_p(S) = \cup_{j \in S} F_p(j)$. Note that (1) $p_j < m_{i,j}$ for all $(i,j) \in F_p$ and (2) if the matching μ with prices p is stable then $\mu \subseteq F_p$.

We define the *feasible first choice graph* $\tilde{G}_p = (I \cup J, \tilde{F}_p)$ at prices p as follows: There is one node per bidder i, one node per item j, and an edge from i to j if and only if item j gives bidder i the highest utility possible, i.e., $u_{i,j}(p_j) \geq u_{i,j'}(p_{j'})$ for all j', *and* $p_j \geq r_{i,j}$. Note that $\tilde{F}_p \subseteq F_p$. For $i \in I$ we define $\tilde{F}_p(i) = \{j : \exists\ (i,j) \in \tilde{F}_p\}$ and similarly $\tilde{F}_p(j) = \{i : \exists\ (i,j) \in \tilde{F}_p\}$. Analogously, for $T \subseteq I$ we define $\tilde{F}_p(T) = \cup_{i \in T} \tilde{F}_p(i)$ and for $S \subseteq J$ we define $\tilde{F}_p(S) = \cup_{j \in S} \tilde{F}_p(i)$. Note that (1) $r_{i,j} \leq p_j < m_{i,j}$ for all $(i,j) \in \tilde{F}_p$ and (2) the matching μ with prices p is stable if and only if $\mu \subseteq \tilde{F}_p$. Also note that the edges in $F_p(i) \setminus \tilde{F}_p(i)$ are all the edges (i,j) with maximum $u_{i,j}(p_j)$ but $p_j < r_{i,j}$.

We define an *alternating path* is a sequence of edges in $\tilde{F}_p$ that alternates between matched and unmatched edges. We require that all but the last item on the path are non-dummy items. The last item can (but does not have to) be the dummy item. A tree in the feasible first choice graph $\tilde{G}_p$ is an *alternating tree* rooted at bidder i if all paths from its root to a leaf are alternating paths that either end with the dummy item, an unmatched item, or a bidder whose feasible first choice items are all contained in the tree. We say that an alternating tree with root i is *maximal* if it is the largest such tree. See Figure 2 for an example.

5. Algorithm

Our algorithm starts with an empty matching and prices all zero. It then matches one bidder after the other by augmenting the current matching along an alternating path. If there is no such path, it repeatedly raises the price of all items in the maximal alternating

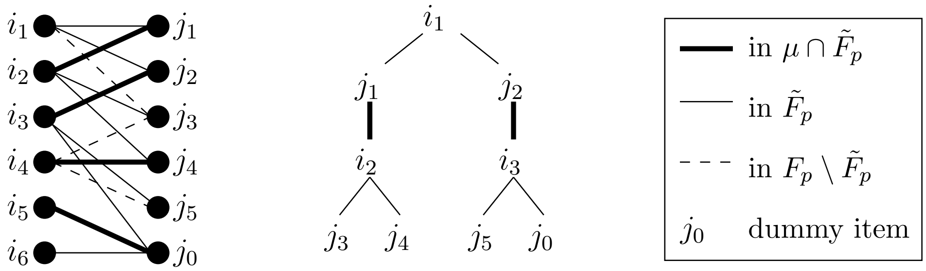


Figure 2: The graph on the left is the (feasible) first choice graph. The bidders i_1 to i_6 are on the left. The items j_1 to j_5 are on the right. The dummy item is j_0. Edges in $\mu \cap \tilde{F}_p$ are thick. Edges in $\tilde{F}_p$ are thin. Edges in $F_p \setminus \tilde{F}_p$ are dashed. The graph on the right is a maximal alternating tree rooted at i_1.

tree under consideration by the minimum amount (a) to make some item $j \notin F_p(i)$ desirable for some bidder i in the tree, or (b) to make some item $j \in F_p(i) \setminus \tilde{F}_p(i)$ feasible for some bidder i in the tree, or (c) to make some item $j \in \tilde{F}_p(i)$ no longer desirable for some bidder i in the tree. Thus it ensures that eventually an alternating path will exist and the matching can be augmented. Note that a matched bidder i can become unmatched if the price of the item j she is matched to reaches $m_{i,j}$. Case (a) corresponds to δ_{out}, Case (b) corresponds to δ_{res}, and Case (c) corresponds to $\delta_{\max}$ in the pseudocode below.

```
Modified Hungarian Method
set $p_j := 0$ for all $j \in J$, $u_i := \max_{j'} v_{i,j'}$ for all $i \in I$, and $\mu := \emptyset$,
while $\exists$ unmatched bidder $i$ do
  find a maximal alternating tree rooted at bidder $i$ in $\tilde{G}_p$
  let $T$ and $S$ be the set of bidders and items in this tree
  while all items $j \in S$ are matched and $j_0 \notin S$ do
    compute $\delta := \min(\delta_{\text{out}}, \delta_{\text{res}}, \delta_{\max})$ where
      $\delta_{\text{out}} := \min_{i \in T, j \notin F_p(i)}(u_i + p_j - v_{i,j})$ [4]
      $\delta_{\text{res}} := \min_{i \in T, j \in F_p(i) \setminus \tilde{F}_p(i)}(r_{i,j} - p_j)$ [4]
      $\delta_{\max} := \min_{i \in T, j \in F_p(i)}(m_{i,j} - p_j)$
    update prices, utilities, and matching by setting
      $p_j := p_j + \delta$ for all $j \in F_p(T)$ \\ leads to a new graph $\tilde{G}_p$
      $u_i := \max_{j'}(v_{i,j'} - p_{j'})$ for all $i \in I$
      $\mu := \mu \cap \tilde{F}_p$ \\ removes unfeasible edges from $\mu$
    find a maximal alternating tree rooted at bidder $i$ in $\tilde{G}_p$
    let $T$ and $S$ be the set of bidders and items in this tree
  end while
  augment $\mu$ along alternating path rooted at $i$ in $\tilde{G}_p$
end while
output $p$, $u$, and $\mu$
```

[4] We need to define $\min_{i \in T, j \in \emptyset}(\ldots) = \infty$ as we might have $F_p(I) = J$ or $F_p(i) \setminus \tilde{F}_p(i) = \emptyset$.

6. Feasibility and Stability

Theorem 6.1. *The Modified HM finds a feasible and stable matching. It can be implemented to run in $O(nk^3 \log(k))$.*

Proof. The matching μ constructed by the Modified HM is a subset of the feasible first choice graph $\tilde{G}_p$ at all times. Hence it suffices to show that after $O(nk^3 \log(k))$ steps all bidders are matched.

The algorithm consists of two nested loops. We analyze the running time in two steps: (1) The time spent in the outer loop *without* the inner loop (ll. 2–4 and 17–18) and (2) the time spent in the inner loop (ll. 5–16). Note that after each execution of the outer while loop the number of matched bidder increases by one. A matched bidder i can only become unmatched if the price of the item j she is matched to reaches $m_{i,j}$. This can happen only once for each pair (i, j), which implies that each bidder can become at most k times unmatched. Thus, the outer loop is executed at most nk times. Since $|S| \leq k$, it follows that $|T| \leq k$. Thus it is straightforward to implement the outer while loop in time $O(k^2)$.

We call an execution of the inner while loop *special* if (a) right before the start of the execution the outer while loop was executed, (b) in the previous iteration of the inner while loop the maximum price of a pair (i, j) was reached, or (c) the reserve price of a pair (i, j) was reached. As each of these cases can happen at most nk times, there are at most $3nk$ special executions of the inner while loop. Non-special executions increase the number of items in the maximal alternating tree by at least one. Thus there are at most k non-special executions between any two consecutive special executions. We present next a data structure that (1) can be built in time $O(k^2)$ and (2) allows to implement *all* non-special executions of the inner while loop between two consecutive special iterations in time $O(k^2 \log k)$. Thus the total time of the algorithm is $O(nk^3 \log k)$.

Data structure:

(1) Keep a list of all bidders in T and a bit vector of length n where bit i is set to 1 if bidder i belongs currently to T and to 0 otherwise. Keep a list of all items in S and bit vector of length k, where bit j is set of 1 if item j belongs currently to S and to 0 otherwise. Finally also keep a list and a bit vector of length k representing all items in $F_p(T)$.

(2) Keep a heap H_{out} and a value δ_{out}, such that H_{out} stores $x_i + p_j - v_{i,j}$ for all pairs (i, j) with $i \in T$ and $j \notin F_p(i)$ and $\delta_{\text{out}} + x_i$ equals u_i for every $i \in T$. Keep a heap H_{res} and a value δ_{res}, such that H_{res} stores $r_{i,j} - y_j$ for all pairs (i, j) with $i \in T$ and $j \in F_p(i) \setminus \tilde{F}_p(i)$ and $\delta_{\text{res}} + y_j$ equals p_j for every $j \in F_p(i) \setminus \tilde{F}_p(i)$. Keep a heap $H_{\max}$ and a value $\delta_{\max}$, such that $H_{\max}$ stores $m_{i,j} - y_j$ for all pairs (i, j) with $i \in T$ and $j \in F_p(i)$ and $\delta_{\max} + y_j$ equals p_j for every $j \in F_p(i)$.

(3) We also store at each bidder i its current u_i, at each item j its current p_j. Thus given a pair (i, j) we can decide in constant time whether $u_i = v_{i,j} - p_j$, i.e., whether $j \in F_p(i)$. Finally we keep a list of edges in μ.

At the beginning of each special execution of the inner while loop a list of bidders and items currently in T and S are passed in either from the preceding execution of the outer while loop (where T and S are constructed in time $O(k^2)$) or from the previous execution of the inner while loop. Recall that $|S| \leq k$ and thus $|T| \leq k$. Thus we can build the above data structures from scratch in time $O(k^2)$ as follows. To initialize the bit vector for T we use the following approach: At the beginning of the algorithm the vector is once initialized to 0, taking time $O(n)$. Then at the beginning of all but the first special execution of the inner while loop the bit vector is "cleaned" by setting the bit of all elements of T in the previous iteration to 0 using the list of elements of T of the previous iteration. Then the list of elements currently in T is used to set the appropriate bits to 1. This takes time $O(k)$ per special execution. The bit vector of items in S has only k entries and thus is simply initialized to 0 at the beginning of each special execution. Then the list of elements currently in S is used to set the appropriate bits to 1. Given the list of bidders in T we decide in constant time for each pair (i, j) with $i \in T$ into which heap(s) its appropriate values should be inserted. If $j \in F_p(i)$ we also add j to $F_p(T)$ if it is not already in this set update the bit vector and the list. When we have processed all pairs (i, j) with $i \in T$ we build the three heaps in time linear in their size such that all δ values are 0. Since $|S| = k$ we know that $|T| = k$. Thus, the initialization takes time $O(k^2)$.

To implement each iteration of the inner while loop we first perform a find-min operation on all three heaps to determine δ. Then we remove all heap values that equal δ. Afterwards we update the price of all items in $F_p(T)$ using the list of $F_p(T)$. We also update the utility of all items in T as follows. If $\delta \neq \delta_{\max}$ updating the utilities is just a simple subtraction per bidder. If $\delta = \delta_{\max}$, i.e., p_j becomes $m_{i,j}$ for some pair (i, j), then updating u_i requires computing $v_{i,j} - p_j$ for all j and potentially removing the edge (i, j) from μ, which in turn might cut a branch of the alternating tree. Thus, in this case we completely rebuild the alternating tree, including S, T, and $F_p(T)$ from scratch. Note however that this can only happen in a special execution of the inner while loop. If $\delta \neq \delta_{\max}$ the elements removed from the heaps tell us which new edges are added to $\tilde{F}_p(T)$ and which new items to add to $F_p(T)$. The new items in $F_p(T)$ gives a set of items from which we start to augment the alternating tree in breadth first manner. For each new item j, we add to $\tilde{F}_p(T)$ the bidder it is matched to as new bidder to S and to $\tilde{F}_p(T)$. For each new bidder i added to $\tilde{F}_p(T)$ we spend time $O(k)$ to determine its adjacent edges in $F_p(i)$ and insert the suitable values for the pairs (i, j) into the three heaps. This process repeats until no new items and no new bidders are added to $F_p(i)$. During this traversal we also update the bit vectors and lists representing T, S, and $F_p(T)$. Let T_{new} be the set of bidders added to T during an execution of the inner while loop and let r be the number of elements removed from the heaps during the execution. Then the above data structures implement the inner while loop in time $O(r * \log k + |T_{\text{new}}| * k.)$ Now note that during a sequence of non-special executions of the inner while loop between two consecutive special executions bidders are never removed from T and each (i, j) pair with $i \in T$ is added (and thus also removed) at most once from each heap. Thus the total number of heap removals during *all* such non-special executions is $3k^2$ and the total number of elements added to T is k, giving a total running time of $O(k^2 \log k)$ for *all* such non-special executions. Since there are at most $3nk$ special executions, the total time for all inner while loops is $O(nk^3 \log k)$. ■

7. Bidder Optimality

Theorem 7.1. *The Modified HM finds a bidder optimal matching in $O(nk^3 \log(k))$ steps.*

We say that a (possibly empty) set $S \subseteq J$ is *strictly overdemanded* for prices p wrt $T \subseteq I$ if (i) $\tilde{F}_p(T) \subseteq S$ and (ii) $\forall\ R \subseteq S$ and $R \neq \emptyset : |\tilde{F}_p(R) \cap T| > |R|$. Using Hall's Theorem [11] one can show that a feasible and stable matching exists for given prices p if and only if there is no strictly overdemanded set of items S in $\tilde{F}_p$.

The proof strategy is as follows: In Lemma 7.2 we show that a feasible and stable matching μ with prices p is bidder optimal if we have that $p_j \leq p'_j$ for all items j and all feasible and stable matchings μ' with prices p'. Afterwards, in Lemma 7.3, we establish a lower bound on the price increase of strictly overdemanded items. Finally, in Lemma 7.4 we argue that whenever the Modified HM updates the prices it updates the prices according to Lemma 7.3. This completes the proof.

Lemma 7.2. *If the matching μ with prices p is stable and $p_j \leq p'_j$ for all j and all stable matchings μ' with prices p', then the matching μ with prices p is bidder optimal.*

Proof. For a contradiction suppose that there exists a feasible and stable matching μ' with prices p' such that $u'_i > u_i$ for some bidder i. Let j be the item that bidder i is matched to in μ and let j' be the item that bidder i is matched to in μ'. Since $p_{j'} \leq p'_{j'}$ and $p'_{j'} < m_{i,j'}$ we have that $u_{i,j'}(p_{j'}) = v_{i,j'} - p_{j'}$. Since the matching μ with prices p is stable we have that $u_i = u_{i,j}(p_j) = v_{i,j} - p_j \geq u_{i,j'}(p_{j'}) = v_{i,j'} - p_{j'}$. It follows that $u'_i = v_{i,j'} - p'_{j'} > u_i = v_{i,j} - p_j \geq v_{i,j'} - p_{j'}$ and, thus, $p'_{j'} < p_{j'}$. This gives a contradiction. ∎

Lemma 7.3. *Given $p = (p_1, \ldots, p_k)$ let $u_i = \max_j u_{i,j}(p_j)$ for all i. Suppose that $S \subseteq J$ is strictly overdemanded for prices p with respect to $T \subseteq I$ and let $\delta = \min(\delta_{out}, \delta_{res}, \delta_{max})$, where $\delta_{out} = \min_{i \in T, j \notin F_p(i)}(u_i + p_j - v_{i,j})$, $\delta_{res} = \min_{i \in T, j \in F_p(i) \setminus \tilde{F}_p(i)}(r_{i,j} - p_j)$, and $\delta_{max} = \min_{i \in T, j \in F_p(i)}(m_{i,j} - p_j)$. Then, for any stable matching μ' with prices p' with $p'_j \geq p_j$ for all j, we have that $p'_j \geq p_j + \delta$ for all $j \in F_p(T)$.*

Proof. We prove the claim in two steps. In the first step, we show that $p'_j \geq p_j + \delta$ for all $j \in \tilde{F}_p(T)$. In the second step, we show that $p'_j \geq p_j + \delta$ for all $j \in F_p(T) \setminus \tilde{F}_p(T)$.

Step 1. Consider the set of items $A = \{j \in \tilde{F}_p(T) \mid \forall k \in \tilde{F}_p(T) : p'_j - p_j \leq p'_k - p_k\}$ and the set of bidders $B = \tilde{F}_p(A) \cap T$. Assume by contradiction that $\delta' = \min_{j \in \tilde{F}_p(T)}(p'_j - p_j) < \delta$. We show that this implies that $|B| > |A| \geq |\tilde{F}_{p'}(B)|$, which gives a contradiction.

The set of items S is strictly overdemanded for prices p wrt to T and A. Thus, since $A \subseteq S$ and $A \neq \emptyset$, $|B| = |\tilde{F}_p(A) \cap T| > |A|$. Next we show that $A \supseteq \tilde{F}_{p'}(B)$ and, thus, $|A| \geq |\tilde{F}_{p'}(B)|$. It suffices to show that $\tilde{F}_{p'}(i) \setminus A = \emptyset$ for all bidders $i \in B$. For a contradiction suppose that there exists a bidder $i \in B$ and an item $k \in \tilde{F}_{p'}(i) \setminus A$. Recall that we must have (1) $u_{i,k}(p'_k) \geq 0$, (2) $u_{i,k}(p'_k) \geq u_{i,k'}(p'_{k'})$ for all k', and (3) $p_k \geq r_{i,k}$. Recall also that (1)–(3) imply that $r_{i,k} \leq p'_k < m_{i,k}$ and so $u_{i,k}(p'_k) = v_{i,k} - p'_k$.

We know that there exists $j \in A$ such that $j \in \tilde{F}_p(i)$. Since $j \in A$ we have that $p'_j < p_j + \delta \leq m_{i,j}$ and so $u_{i,j}(p'_j) = v_{i,j} - p'_j$. Thus, since $k \in \tilde{F}_{p'}(i)$, $v_{i,k} - p'_k \geq v_{i,j} - p'_j$. Finally, since $j \in \tilde{F}_p(i)$ and $p_k \leq p'_k < m_{i,k}$, we have that $u_{i,j}(p_j) = v_{i,j} - p_j \geq u_{i,k}(p_k) = v_{i,k} - p_k$.

Case 1: $k \in J \setminus F_p(B)$. Since $\delta \leq \delta_{\text{out}} \leq u_i + p_k - v_{i,k}$ and $u_i = v_{i,j} - p_j$ we have that $\delta \leq v_{i,j} - p_j + p_k - v_{i,k}$. Rearranging this gives $v_{i,k} - p_k + \delta \leq v_{i,j} - p_j$. Since $p'_k \geq p_k$ and $p_j > p'_j - \delta$ this implies that $v_{i,k} - p'_k < v_{i,j} - p'_j$. Contradiction!

Case 2: $k \in F_p(B) \setminus \tilde{F}_p(B)$. If $p'_k - p_k \leq p'_j - p_j = \delta'$ then $p'_k \leq p_k + \delta' < p_k + \delta$. Since $\delta \leq \delta_{\text{res}} \leq r_{i,k} - p_k$ this implies that $p'_k < r_{i,k}$. Contradiction! Otherwise, $p'_k - p_k > p'_j - p_j$. Since $v_{i,j} - p_j \geq v_{i,k} - p_k$ this implies that $v_{i,j} - p'_j > v_{i,k} - p'_k$. Contradiction!

Case 3: $k \in \tilde{F}_p(B) \setminus A$. Since $j \in A$ and $k \notin A$ we have that $p'_k - p_k > \delta' = p'_j - p_j$. Since $v_{i,j} - p_j \geq v_{i,k} - p_k$ this implies that $v_{i,j} - p'_j > v_{i,k} - p'_k$. Contradiction!

Step 2. Consider an arbitrary item $j \in F_p(T) \setminus \tilde{F}_p(T)$ such that $p'_j - p_j \leq p'_{j'} - p_{j'}$ for all $j' \in F_p(T) \setminus \tilde{F}_p(T)$ and a bidder $i \in T$ such that $j \in F_p(i)$. Assume by contradiction that $\delta' = p'_j - p_j < \delta$. We show that this implies that $\tilde{F}_{p'}(i) = \emptyset$, which gives a contradiction.

First observe that $\delta' < \delta \leq \delta_{\text{res}} \leq r_{i,j} - p_j$ and, thus, $p'_j < p_j + \delta \leq r_{i,j}$, which shows that $j \notin \tilde{F}_{p'}(i)$. Next consider an arbitrary item $k \neq j$. For a contradiction suppose that $k \in \tilde{F}_{p'}(i)$. It follows that $r_{i,k} \leq p'_k < m_{i,k}$ and $u_{i,k}(p'_k) = v_{i,k} - p'_k \geq u_{i,j}(p'_j)$.

Since $p'_j = p_j + \delta' < p_j + \delta \leq m_{i,j}$ we have that $u_{i,j}(p'_j) = v_{i,j} - p'_j$ and so $v_{i,k} - p'_k \geq v_{i,j} - p'_j$. Finally, since $j \in F_p(i)$ and $p_k \leq p'_k < m_{i,k}$, we have that $u_{i,j}(p_j) = v_{i,j} - p_j \geq u_{i,k}(p_k) = v_{i,k} - p_k$.

Case 1: $k \in J \setminus F_p(T)$. Since $\delta \leq \delta_{\text{out}} \leq u_i + p_k - v_{i,k}$ and $u_i = v_{i,j} - p_j$ we have that $\delta \leq v_{i,j} - p_j + p_k - v_{i,k}$. Rearranging this gives $v_{i,k} - p_k + \delta \leq v_{i,j} - p_j$. Since $p'_k \geq p_k$ and $p_j > p'_j - \delta$ this implies that $v_{i,k} - p'_k < v_{i,j} - p'_j$. Contradiction!

Case 2: $k \in F_p(T) \setminus \tilde{F}_p(T)$. If $p'_k - p_k \leq p'_j - p_j = \delta'$ then $p'_k \leq p_k + \delta' < p_k + \delta$. Since $\delta \leq \delta_{\text{res}} \leq r_{i,k} - p_k$ this implies that $p'_k < r_{i,k}$. Contradiction! Otherwise, $p'_k - p_k > p'_j - p_j$. Since $v_{i,j} - p_j \geq v_{i,k} - p_k$ this implies that $v_{i,j} - p'_j > v_{i,k} - p'_k$. Contradiction!

Case 3: $k \in \tilde{F}_p(T)$. From Step 1 we know that $p'_k - p_k \geq \delta > \delta' = p'_j - p_j$. Since $v_{i,j} - p_j \geq v_{i,k} - p_k$ this implies that $v_{i,j} - p'_j > v_{i,k} - p'_k$. Contradiction! ∎

Lemma 7.4. *Let p be the prices computed by the Modified HM. Then for any stable matching μ' with prices p' we have that $p_j \leq p'_j$ for all j.*

Proof. We prove the claim by induction over the price updates. Let p^t denote the prices after the t-th price update.

For $t = 0$ the claim follows from the fact that $p^t = 0$ and $p'_j \geq 0$ for all items j and all feasible matchings μ' with prices p'.

For $t > 0$ assume that the claim is true for $t - 1$. Let S be the set of items and let T be the set of bidders considered by the matching mechanism for the t-th price update. We claim that S is strictly overdemanded for prices p^{t-1} wrt to T. This is true because: (1) S and T are defined as the set of items resp. bidders in a *maximal* alternating tree and, thus, there are no edges in $\tilde{F}_{p^{t-1}}$ from bidders in T to items in $J \setminus S$ which shows that $\tilde{F}_{p^{t-1}}(T) \subseteq S$. (2) For all subsets $R \subset S$ and $R \neq \emptyset$ the number of "neighbors" in the alternating tree under consideration is strictly larger than $|R|$ which shows that $|\tilde{F}_{p^{t-1}}(R) \cap T| > |R|$. By the induction hypothesis $p'_j \geq p^{t-1}_j$ for all items $j \in J$ and, thus, Lemma 7.3 shows that $p'_j \geq p^{t-1}_j + \delta$ for all items $j \in F_{p^{t-1}}(t)$. The Modified HM sets $p^t_j = p^{t-1}_j + \delta$ for all items $j \in F_{p^{t-1}}(T)$ and $p^t_j = p^{t-1}_j$ for all items $j \notin F_{p^{t-1}}(T)$ and so $p'_j \geq p^t_j$ for *all* items $j \in J$. ∎

8. Truthfulness

The following example shows that with our notion of stability bidder optimality no longer implies truthfulness, even if (i) there are no reserve prices, i.e., $r_{i,j} = 0$ for all i and j, (ii) maximum prices depend only on the item, i.e., for all i there exists a constant m_i such that $m_{i,j} = m_i$ for all j, and (iii) no two bidders have the same maximum price, i.e., $m_i \neq m_k$ for any two bidders $i \neq k$. More specifically, it shows that a bidder can improve her utility by lying about the valuation of a single item. Since the bidder optimal utilities are uniquely defined, this shows that no mechanism that computes a bidder optimal matching μ with prices p can be truthful. Note that if (i) to (iii) hold and there exists constants

$\alpha_1 \geq \cdots \geq \alpha_k$ and $v_1, \ldots, v_k$ such that $v_{i,j} = v_i \cdot \alpha_j$ for all i and j, then Ashlagi et al. [3] show the existence of a truthful mechanism.

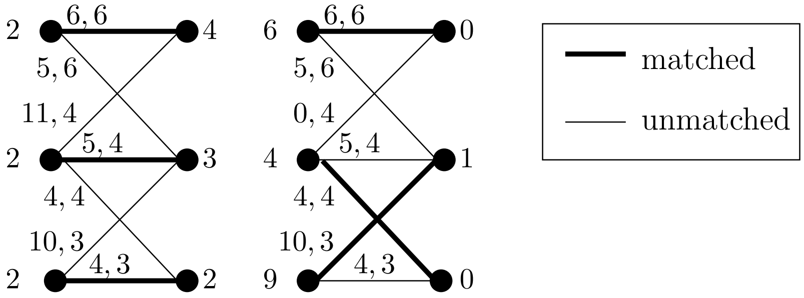


Figure 3: Bidders are on the left and items are on the right. The numbers next to the bidders indicate their utilities. The numbers next to the items indicate their prices. The labels along the edges show valuations and maximum prices. The graph on the left depicts the bidder optimal matching for the "true" valuations. The graph on the right depicts the bidder optimal matching for the "falsified" valuations. Specifically, in the matching on the right bidder 2 misreports her valuation for item 1. This gives her a strictly higher utility, and shows that lying "pays off".

9. Generalized Linear Utilities

The following theorem generalizes our results to utilities of the form $u_{i,j}(p_j) = v_{i,j} - c_i \cdot c_j \cdot p_j$ for $p_j < m_{i,j}$ and $u_{i,j}(p_j) = -\infty$ otherwise. This reduction does *not* work if $u_{i,j}(p_j) = v_{i,j} - c_{i,j} \cdot p_j$ for $p_j < m_{i,j}$ and $u_{i,j}(p_j) = -\infty$ otherwise. We prove the existence of a bidder optimal solution for more general utilities in [6].

Theorem 9.1. *The matching $\hat{\mu}$ with prices $\hat{p}$ is bidder optimal for $\hat{v} = (\hat{v}_{i,j})$, $\hat{r} = (\hat{r}_{i,j})$, $\hat{m} = (\hat{m}_{i,j})$ and utilities $u_{i,j}(p_j) = v_{i,j} - c_i \cdot c_j \cdot p_j$ if $p_j < m_{i,j}$ and $u_{i,j}(p_j) = -\infty$ otherwise if and only if the matching μ with prices p, where $\mu = \hat{\mu}$ and $p = (c_j \cdot \hat{p}_j)$, is bidder optimal for $v = (\hat{v}_{i,j}/c_i)$, $r = (c_j \cdot \hat{r}_{i,j})$, $m = (c_j \cdot \hat{m}_{i,j})$ and utilities $u_{i,j}(p_j) = v_{i,j} - p_j$ if $p_j < m_{i,j}$ and $u_{i,j}(p_j) = -\infty$ otherwise.*

Proof. Since $\hat{p}_j < \hat{m}_{i,j}$ if and only if $p < m_{i,j}$ we have that $\hat{u}_{i,j}(\hat{p}_j) = c_i \cdot u_{i,j}(p_j)$. Since $\hat{\mu} = \mu$ this implies that $\hat{u}_i = c_i \cdot u_i$ for all i.

Feasibility. Since $c_i > 0$ for all i we have that $\hat{u}_i \geq 0$ for all i if and only if $u_i = \hat{u}_i/c_i \geq 0$ for all i. Since $c_j > 0$ for all i we have that $\hat{p}_j \geq 0$ for all j if and only if $p_j = c_j \cdot \hat{p}_j \geq 0$ for all j. Since $\mu = \hat{\mu}$ and $r_{i,j} = c_j \cdot \hat{r}_{i,j}$, $p_j = c_j \cdot \hat{p}_j$, and $m_{i,j} = c_j \cdot \hat{m}_{i,j}$ for all i and j we have that $\hat{r}_{i,j} \leq \hat{p}_j < \hat{m}_{i,j}$ for all $(i,j) \in \hat{\mu}$ if and only if $r_{i,j} \leq p_j < m_{i,j}$ for all $(i,j) \in \mu$.

Stability. If $\hat{\mu}$ with $\hat{p}$ is stable then μ with p is stable because $u_i = c_i \cdot \hat{u}_i \geq c_i \cdot \hat{u}_{i,j}(\hat{p}_j) = u_{i,j}(p_j)$ for all i and j. If μ with p is stable then $\hat{\mu}$ with $\hat{p}$ is stable because $\hat{u}_i = u_i/c_i \geq u_{i,j}(p_j)/c_i = \hat{u}_{i,j}(\hat{p}_j)$ for all i and j.

Bidder Optimality. For a contraction suppose that $\hat{\mu}$ with $\hat{p}$ is bidder optimal but μ with p is not. Then there must be a feasible and stable matching μ' with p' such that $u'_i > u_i$ for at least one bidder i. By transforming μ' with p' into $\hat{\mu}'$ with $\hat{p}'$ we get a feasible and stable matching for which $\hat{u}'_i = c_i \cdot u'_i > c_i \cdot u_i = \hat{u}_i$. Contradiction!

For a contraction suppose that μ with p is bidder optimal but $\hat{\mu}$ with $\hat{p}$ is not. Then there must be a feasible and stable matching $\hat{\mu}'$ with $\hat{p}'$ such that $\hat{u}_i' > \hat{u}_i$ for at least one bidder i. By transforming $\hat{\mu}'$ with $\hat{p}'$ into μ' with p' we get a feasible and stable matching for which $u_i' = \hat{u}_i'/c_i > \hat{u}_i/c_i = u_i$. Contradiction! ■

References

[1] G. Aggarwal, A. Goel, and R. Motwani. Truthful auctions for pricing search keywords. *Proceedings of the Conference on Electronic Commerce*, pages 1–7, 2006.

[2] G. Aggarwal, S. Muthukrishnan, D. Pál, and M. Pál. General auction mechanism for search advertising. *Proceedings of the World Wide Web Conference*, pages 241–250, 2009.

[3] I. Ashlagi, M. Braverman, A. Hassidim, R. Lavi, and M. Tennenholtz. Position auctions with budgets: Existence and uniqueness. *Working Paper*, 2009.

[4] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[5] G. Demange, D. Gale, and M. Sotomayor. Multi-item auctions. *Political Economy*, 94(4):863–72, 1986.

[6] P. Dütting, M. Henzinger, and I. Weber. Bidder optimal assignments for general utilities. *Proceedings of the Workshop on Internet and Network Economics*, pages 575–582, 2009.

[7] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97(1):242–259, 2007.

[8] A. Frank. On Kuhn's Hungarian Method. *Naval Research Logistics*, 51:2–5, 2004.

[9] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[10] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[11] P. Hall. On representatives of subsets. *London Mathematical Society*, 10:26–30, 1935.

[12] D. Kempe, A. Mu'alem, and M. Salek. Envy-free allocations for budgeted bidders. *Proceedings of the Workshop on Internet and Network Economics*, pages 537–544, 2009.

[13] H. W. Kuhn. The Hungarian Method for the assignment problem. *Naval Research Logistics*, 2:83–97, 1955.

[14] S. Lahaie, D. M. Pennock, A. Saberi, and R. V. Vohra. *Algorithmic Game Theory*, chapter 28, pages 699–716. Cambridge University Press, 2007.

[15] A. E. Roth and M. Sotomayor. *Two-sided matching: A study in game-theoretic modeling and analyis.* Cambridge University Press, 1990.

[16] L. S. Shapley and M. Shubik. The assignment game: The core I. *Game Theory*, 29:111–130, 1972.

[17] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Finance*, 16:8–27, 1961.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 299-310
www.stacs-conf.org

DISPERSION IN UNIT DISKS

ADRIAN DUMITRESCU [1] AND MINGHUI JIANG [2]

[1] Department of Computer Science, University of Wisconsin–Milwaukee, WI 53201-0784, USA
E-mail address: ad@cs.uwm.edu

[2] Department of Computer Science, Utah State University, Logan, UT 84322-4205, USA
E-mail address: mjiang@cc.usu.edu

Abstract. We present two new approximation algorithms with (improved) constant ratios for selecting n points in n unit disks such that the minimum pairwise distance among the points is maximized.

(I) A very simple $O(n \log n)$-time algorithm with ratio 0.5110 for disjoint unit disks. In combination with an algorithm of Cabello [3], it yields a $O(n^2)$-time algorithm with ratio of 0.4487 for dispersion in n not necessarily disjoint unit disks.

(II) A more sophisticated LP-based algorithm with ratio 0.6495 for disjoint unit disks that uses a linear number of variables and constraints, and runs in polynomial time. The algorithm introduces a novel technique which combines linear programming and projections for approximating distances.

The previous best approximation ratio for disjoint unit disks was $\frac{1}{2}$. Our results give a partial answer to an open question raised by Cabello [3], who asked whether $\frac{1}{2}$ could be improved.

1. Introduction

Let $\mathcal{R}$ be a family of n subsets of a metric space. The problem of *dispersion in* $\mathcal{R}$ is that of selecting n points, one in each subset, such that the minimum inter-point distance is maximized. This dispersion problem was introduced by Fiala et al. [6] as "systems of distant representatives", generalizing the classic problem "systems of distinct representatives". An especially interesting version of the dispersion problem, which has natural applications to wireless networking and map labeling, is in a geometric setting where $\mathcal{R}$ is a set of unit disks in the plane.

Fiala et al. [6] showed that dispersion in (not necessarily disjoint) unit disks is NP-hard. It is not difficult to modify their construction, which gives a reduction from Planar-3SAT, to show that dispersion in disjoint unit disks is also NP-hard. Moreover, by a slackness argument [7, 8], the same construction also implies that the problem is APX-hard; i.e, unless

1998 ACM Subject Classification: F.2.2 Geometrical problems and computations.

Key words and phrases: Dispersion problem, linear programming, approximation algorithm.

Adrian Dumitrescu was supported in part by NSF CAREER grant CCF-0444188; part of the research by this author was done at Ecole Polytechnique Fédérale de Lausanne. Minghui Jiang was supported in part by NSF grant DBI-0743670.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2464

P = NP, the problem does not admit any polynomial-time approximation scheme. On the positive side, Cabello [3] presented a quadratic-time approximation algorithm with ratio $0.4465\ldots (1/2.2393\ldots)$ for dispersion in not necessarily disjoint unit disks. For dispersion in disjoint unit disks, Cabello [3] noticed that a naive algorithm called Centers, which simply selects the centers of the given disks as the points, gives a $\frac{1}{2}$-approximation.

We first introduce some preliminaries. For two points, $p = (x_p, y_p)$ and $q = (x_q, y_q)$, let $|pq|$ denote the Euclidean distance between them: $|pq| = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$. A *unit disk* is a disk of radius one. Let the *distance between two disks* be the distance between their centers; e.g., the distance between two tangent disks is 2. Let $\mathcal{D}$ be a set of n disjoint unit disks in the plane. Let δ be the minimum pairwise distance of the disks in $\mathcal{D}$; clearly $\delta \geq 2$. The algorithm Centers, by the obvious inequalities APX $\geq \delta$ and OPT $\leq \delta + 2$, achieves an approximation ratio

$$\frac{\mathrm{APX}}{\mathrm{OPT}} \geq \frac{\delta}{\delta + 2} \geq \frac{1}{2}.$$

Observe that the approximation ratio of Centers gets better as δ increases; in fact, it can get arbitrarily close to 1, if δ is large enough. Cabello asked whether this trivial $\frac{1}{2}$-approximation can be improved for disjoint unit disks [3, p. 72].

We start with a very simple and efficient algorithm that achieves a ratio better than $\frac{1}{2}$ for dispersion in disjoint unit disks, and a ratio slightly better than 0.4465 for dispersion in not necessarily disjoint unit disks:

Theorem 1.1. *There is an $O(n \log n)$-time approximation algorithm with ratio* 0.5110 *for dispersion in n disjoint unit disks. In combination with an algorithm of Cabello, it yields a $O(n^2)$-time algorithm with ratio of* 0.4487 *for dispersion in n not necessarily disjoint unit disks.*

Using linear programming, we then obtain the following substantially better approximation for dispersion in disjoint unit disks:

Theorem 1.2. *There is an LP-based approximation algorithm, with $O(n)$ variables and constraints, and running in polynomial time, that achieves approximation ratio* 0.6495*, for dispersion in n disjoint unit disks.*

It is likely that our method for proving Theorem 1.2, which uses projections for approximating distances, and linear programming for optimization, is also applicable to other optimization problems involving distances.

Related work. The problem studied in this paper, dispersion in unit disks, is related to a few other problems in computational geometry. We mention three results that are more closely related to ours:

(1) For labeling n points with n disjoint congruent disks, each point on the boundary of a distinct disk, such that radius of the disks is maximized, Jiang et al. [8] presented a $\frac{1}{2.98+\varepsilon}$-approximation algorithm, and proved that the problem is NP-hard to approximate with ratio more than $\frac{1}{1.0349}$.

(2) For packing of n axis-parallel congruent squares (congruent disks in the L_∞ metric) in the same rectilinear polygon such that the side length of the squares is maximized, Baur and Fekete [1] presented a $\frac{2}{3}$-approximation algorithm, and proved that the problem is NP-hard to approximate with ratio more than $\frac{13}{14}$.

(3) A $\frac{2}{3}$-approximation algorithm for a related problem of packing n unit disks in a rectangle without overlapping an existent set of m unit disks in the same rectangle, has been obtained by Benkert et al. [2].

(4) Given n points in the plane, Demaine et al. [4] considered the problem of moving them to an *independent set* in the unit disk graph metric: that is, each point has to move to a position such that all pairwise distances are at least 1, and such that the maximum distance a point moved is minimized. They presented an approximation algorithm, which achieves a good ratio if the points are initially "far from" an independent set. However the approximation ratio becomes unbounded for instances that are "very close to" an independent set. Observe that in this problem, the optimum may be arbitrarily small, i.e., arbitrarily close to 0.

2. A simple approximation algorithm for unit disks

In this section we present a very simple approximation algorithm **A1** for dispersion in (not necessarily disjoint) unit disks, and prove Theorem 1.1. The idea of the algorithm is as follows. Recall that δ is the minimum pairwise distance among the unit disks. Let $\sigma = \sigma(\delta)$ be a positive parameter to be specified; in particular, at the threshold distance $\delta = 2$ for disjoint unit disks, we have $\sigma(2) = 2.0883\ldots$, which is only slightly larger than δ. Consider the *distance graph* of the unit disks for the parameter σ, which has a vertex for each disk, and an edge between two vertices if and only if the corresponding disks have distance at most σ. If there is a vertex of degree at least two in the distance graph, that is, if there is a disk close to two other disks, then a packing argument shows that the minimum pairwise distance of any three points in the three disks must be small. Thus simply placing the points at the disk centers already achieves a good approximation ratio. Otherwise, every vertex in the distance graph has degree at most one, and the edges form a matching. In this case, the disks that are close to each other are grouped into pairs. The distance between the two points in each pair can be slightly increased by moving them away from the disk centers, at the cost of possibly decreasing the distances between points in different pairs.

Let $\mathcal{D}$ be a set of n (not necessarily disjoint) unit disks in the plane. The algorithm **A1** consists of three steps:

1. Compute the minimum pairwise distance δ of the disks in $\mathcal{D}$, and for each disk, find the two disks closest to it.
2. If the distance from some disk to its second closest disk is at most $\sigma = \sigma(\delta)$, return the n disk centers as the set of points. Otherwise, proceed to the next step.
3. Place a point at the center of each disk. Then, for each disk, if the distance from the disk to its closest disk is at most σ, move the point away from the closest disk for a distance of $(\sigma - \delta)/4$, so that the two points in each close pair of disks are moved in opposite directions; we will show that $\delta < \sigma < \delta + 4$, thus the distance $(\sigma - \delta)/4$ is between 0 and 1, and each point remains in its own disk. Finally, return the set of points.

Algorithm analysis. The bottleneck for the running time of the algorithm **A1** is simply the computation of the two closest disks from each disk in step 1, which takes $O(n \log n)$ time [5, p. 306]. The other two steps of the algorithm can clearly be done in $O(n)$ time.

For the proof of the approximation ratio, define the following function $f(s)$ for $s \geq 0$:

$$f(s) = \sqrt{(1+s)^2 + 1/2 + \sqrt{3(1+s)^2 - 3/4}}. \tag{2.1}$$

The function $f(\cdot)$ is increasing and $f(0) = \sqrt{3}$. The justification for step 2 of the algorithm **A1** is the following packing lemma (its proof is omitted). Here the disk with center O is close to two other disks with centers P and Q, respectively; see Figure 1.

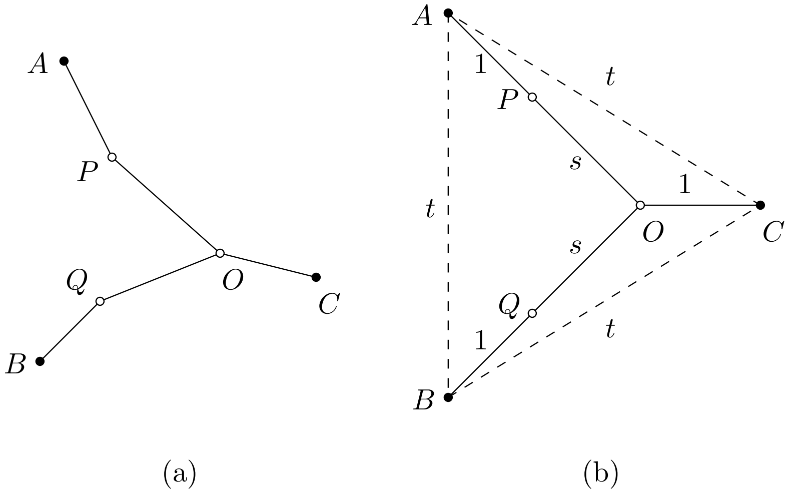


Figure 1: (a) A linkage of the five segments AP, BQ, CO, OP, OQ for three points A, B, C in three unit disks with centers P, Q, O, respectively. (b) The extreme configuration: A, P, O are collinear, B, Q, O are collinear, $|AP| = |BQ| = |CO| = 1$, $|OP| = |OQ| = s$, $|AC| = |BC| = |AB| = t$.

Lemma 2.1. *Let A, B, C be three points in three unit disks with centers P, Q, O, respectively. Let $s = \max\{|OP|, |OQ|\}$ and $t = \min\{|AC|, |BC|, |AB|\}$. Then $t \leq f(s)$.*

Consider the following equation in σ:

$$\frac{\delta}{f(\sigma)} = \frac{\sigma + \delta}{2(\delta + 2)}. \tag{2.2}$$

The next lemma (its proof is omitted) confirms that σ exists and lies in the desired range:

Lemma 2.2. *There is a unique solution σ to (2.2). Moreover, $\delta < \sigma < \delta + 4$.*

We now analyze the approximation ratio of the algorithm **A1**. Let APX be the minimum pairwise distance of the points returned by the algorithm. Let OPT be the minimum pairwise distance of the optimal set of points. Let

$$c = c(\delta) = \frac{\delta}{f(\sigma)} = \frac{\sigma + \delta}{2(\delta + 2)}. \tag{2.3}$$

We next prove that APX $\geq c \cdot$ OPT by considering two cases:

- If the algorithm returns the n disk centers as the set of points in step 2, then there is a disk such that the distances from the disk to its two closest disks are at most σ. By Lemma 2.1, we have OPT $\leq f(\sigma)$. Since APX $= \delta$, it follows that

$$\frac{\text{APX}}{\text{OPT}} \geq \frac{\delta}{f(\sigma)}. \tag{2.4}$$

- If the algorithm proceeds to step 3, then the distance from each disk to its second closest disk is more than σ. If two disks have distance at most σ, then they must be the closest disks of each other, and the movements of points in step 3 ensure that their two points have distance at least $\delta + 2(\sigma - \delta)/4 = (\sigma + \delta)/2$. On the other hand, if two disks have distance more than σ, then after the movements their two points have distance at least $\sigma - 2(\sigma - \delta)/4 = (\sigma + \delta)/2$. Thus $\text{APX} \geq (\sigma + \delta)/2$. Since $\text{OPT} \leq \delta + 2$, it follows that
$$\frac{\text{APX}}{\text{OPT}} \geq \frac{\sigma + \delta}{2(\delta + 2)}. \tag{2.5}$$

By (2.3), (2.4), and (2.5), the algorithm **A1** achieves an approximation ratio of $c(\delta)$ for $\delta \geq 0$. It can be verified that $c(\delta)$ is an increasing function of δ for $\delta \geq 0$. Thus, for dispersion in disjoint unit disks, the approximation ratio is
$$c(\delta) \geq c(2) = 0.5110\ldots, \quad \text{for } \delta \geq 2.$$

For dispersion in not necessarily disjoint unit disks, Cabello [3] presented a hybrid algorithm that applies two different algorithms Placement and Centers then returns the better solution. We now briefly review Cabello's analysis for the hybrid algorithm. Let $x = \text{OPT}/2$ (the scaling here is necessary because Cabello defined a unit disk as a disk of unit diameter instead of unit radius). The algorithm Placement, which runs in $O(n^2)$ time, achieves a ratio of
$$c_1(x) = \frac{-\sqrt{3} + \sqrt{3}x + \sqrt{3 + 2x - x^2}}{4x}, \quad \text{for } 1 \leq x \leq 2,$$
and a ratio of at least $\frac{1}{2}$ for $0 \leq x \leq 1$. The algorithm Centers achieves a ratio of
$$c_2(x) = \frac{x - 1}{x}, \quad \text{for } x \geq 1,$$
which is at least $\frac{1}{2}$ for $x \geq 2$. Refer to Figure 2. Since $c_1(x)$ is decreasing in x and $c_2(x)$ is increasing in x, the minimum approximation ratio of the hybrid algorithm occurs at the intersection of the two curves $c_1(x)$ and $c_2(x)$ for $1 \leq x \leq 2$: precisely, $c_1(x) = c_2(x) = 0.4465\ldots(1/2.2393\ldots)$ for $x = 1.8068\ldots$.

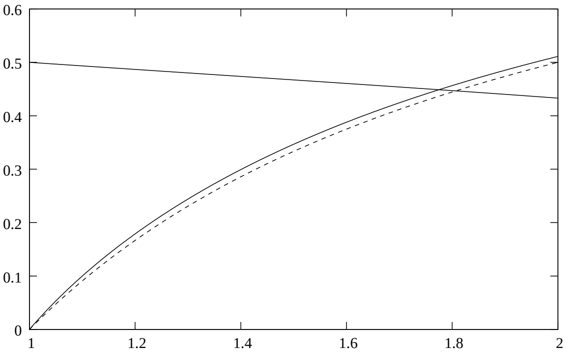


Figure 2: Approximation ratios $c_1(x)$, $c_2(x)$, and $c_3(x)$ for $1 \leq x \leq 2$. The solid decreasing curve is $c_1(x)$. The dashed increasing curve is $c_2(x)$. The solid increasing curve is $c_3(x)$.

Now define
$$c_3(x) = c(2x - 2), \quad \text{for } x \geq 1.$$

From the obvious inequality $\mathrm{OPT} \leq \delta + 2$, we have $\delta \geq \mathrm{OPT} - 2 = 2x - 2$. Recall that the function $c(\delta)$ is increasing in δ. Thus our algorithm **A1** achieves an approximation ratio of at least $c(\delta) \geq c(2x - 2) = c_3(x)$ for $x \geq 1$. It can be verified that $c_2(x) = c_3(x) = 0$ for $x = 1$ and $0 < c_2(x) < c_3(x) < 1$ for $x > 1$. Refer back to Figure 2. Replace the algorithm CENTERS by our algorithm **A1** in the hybrid algorithm. Then the two curves $c_1(x)$ and $c_3(x)$ intersects at $x = 1.7750\dots$ and, correspondingly, the minimum approximation ratio of the new hybrid algorithm is $0.4487\dots$ $(1/2.2284\dots)$. This completes the proof of Theorem 1.1.

3. An LP-based approximation algorithm for disjoint unit disks

In this section we present and analyze approximation algorithm **A2**. We first introduce some definitions and notations. Let $\Omega_1, \dots, \Omega_n$ be n pairwise disjoint unit disks, and let o_i be the center of Ω_i. Denote by δ the minimum pairwise distance among the disks; clearly, $\delta \geq 2$. The algorithm computes δ in $O(n \log n)$ time in a preliminary step.

Let $r = r(\delta)$, where $0 < r \leq 1$, be a parameter that will be chosen later, in order to maximize the approximation ratio. For $i = 1, \dots, n$, let $\omega_i \subset \Omega_i$ be a concentric disk of radius r. Let $\alpha_{ij} \in [-\pi/2, \pi/2)$ be the direction (or angle) of the line determined by o_i and o_j. For $\alpha \in [-\pi/2, \pi/2)$, let ℓ_α be any line of direction α. For two vectors $\overline{u} = (u_1, u_2)$, and $\overline{v} = (v_1, v_2)$, their dot product is $\langle \overline{u} \cdot \overline{v} \rangle = u_1 v_1 + u_2 v_2$. The scalar projection of $\overline{v}$ onto $\overline{u}$ is given by the formula

$$\mathrm{proj}_{\overline{u}} \overline{v} = \frac{\langle \overline{u} \cdot \overline{v} \rangle}{|\overline{u}|}. \tag{3.1}$$

For two points, p and q, let $\mathrm{proj}_\alpha(p, q)$ denote the length of the projection of the segment pq onto a line ℓ_α of direction α, i.e., onto the vector $(\cos \alpha, \sin \alpha)$.

Our approximation algorithm can be viewed as a two step process: STEP 1. We first restrict the feasible region of each point p_i, from the given unit disk Ω_i to a smaller concentric disk ω_i of radius r, $0 < r < 1$. Further, we approximate each smaller disk ω_i by an inscribed regular polygon with sufficiently many sides (say, 64). For convenience however, we still use "disks" when referring to the convex polygons approximating (inscribed in) the smaller disks. Note that this first step is only conceptual. STEP 2. We find a good approximation for the dispersion problem constrained to the smaller size disks.

The idea is as follows: Observe that after STEP 1, the centers of the original disks Ω_i are still in the feasible regions for each of the n points. So the $\frac{1}{2}$ approximation that we could easily achieve earlier, is still attainable. Secondly, observe that if r is sufficiently small, then the distance between two points (in two smaller disks) can be well approximated by the projection of the segment connecting the two points onto the line connecting the centers of the two disks. The length of each such projection can be expressed as a linear combination of the coordinates of the two points, and we can use linear programming in order to maximize the smallest projection length of an inter-point distance. So all the constraints in the dispersion problem will be expressed as linear inequalities, at the cost of finding only an approximate solution. The resulting approximation ratio of the algorithm is the product of the ratios achievable in STEP 1 and STEP 2. In the end, we select r so as to maximize the overall ratio. We now present the technical details.

We start with a technical lemma that guarantees that a large fraction of the distance between two points in two smaller disks is preserved by projection onto the line through the two disk centers (STEP 2).

Lemma 3.1. *Let ω_i, ω_j be two congruent disjoint disks of radius r, where $0 < r \leq 1$, at distance $d \geq \delta \geq 2$. Let ℓ_{ij} be the line determined by o_i and o_j, and ℓ be a line that intersects both ω_i and ω_j. Let α be the (nonnegative) angle between ℓ_{ij} and ℓ. Then $\cos\alpha \geq \frac{\sqrt{d^2-4r^2}}{d} \geq \frac{\sqrt{\delta^2-4r^2}}{\delta}$.*

Proof. We can assume w.l.o.g. that ℓ_{ij} is horizontal; see Figure 3. By symmetry, we can assume that ℓ has positive slope. We claim that if $\alpha \in [0, \pi/2]$ is maximized, then ℓ must be tangent to ω_i and ω_j. Assume for instance that ℓ is not tangent to ω_j, as illustrated in the figure. Select a point p on ℓ left of the intersections points of ℓ with $\partial\omega_i$, and $\partial\omega_j$, and rotate ℓ counterclockwise around p until ℓ becomes tangent to ω_j. The angle α increases in this operation, a contradiction of the assumed maximality. We conclude that ℓ must be tangent to ω_i and ω_j in the first place, as desired. The angle formula $\cos\alpha = \frac{\sqrt{d^2-4r^2}}{d}$ is now easily verified to hold in the tangent case. ■

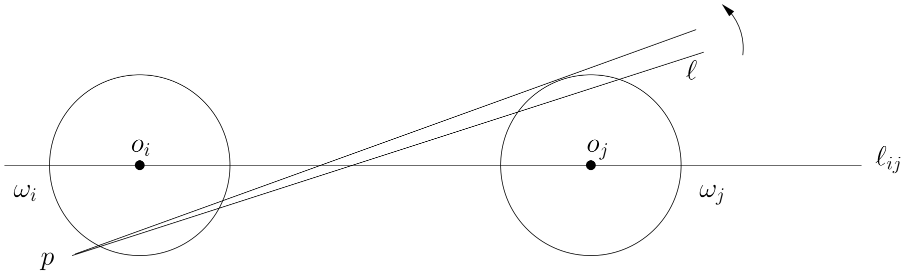


Figure 3: Lemma 3.1.

The next two lemmas guarantee that a large fraction of OPT survives after restricting the feasible regions to smaller disks (Step 1).

Lemma 3.2. *Consider two disjoint unit disks Ω_i and Ω_j at distance $|o_io_j| = d$. Let $p_i \in \Omega_i$ and $p_j \in \Omega_j$ be two points. Let $q_i \in \omega_i$ be the point on o_ip_i at distance $r|o_ip_i|$ from o_i. Similarly define $q_j \in \omega_j$ as the point on o_jp_j at distance $r|o_jp_j|$ from o_j. Then*

$$\frac{|q_iq_j|}{|p_ip_j|} \geq \frac{d+2r}{d+2}. \tag{3.2}$$

This inequality is tight.

Proof. We can assume w.l.o.g. that $o_i = (0, 0)$ and $o_j = (d, 0)$, where $d \geq 2$. To represent points, we use complex numbers in the proof. The point p_i is represented by z_1, where $z_1 \in \mathbb{C}$, with $|z_1| \leq 1$; hence q_i is represented by rz_1. The point p_j is represented by $d + z_2$, where $z_2 \in \mathbb{C}$, with $|z_2| \leq 1$; hence q_j is represented by $d + rz_2$. With this notation, the claimed inequality is

$$\frac{|d + rz_2 - rz_1|}{|d + z_2 - z_1|} \geq \frac{d+2r}{d+2}. \tag{3.3}$$

Write $z = z_2 - z_1$, and note that $|z| \leq |z_1| + |z_2| \leq 2$. Inequality (3.3) can be written now as

$$\frac{|d + rz|}{|d + z|} \geq \frac{d+2r}{d+2}. \tag{3.4}$$

Let $z = a(\cos\alpha + i\sin\alpha)$, be the complex number representation of z, where $0 \le a \le 2$, and $\alpha \in [0, 2\pi]$. We have

$$|d + z|^2 = (a\cos\alpha + d)^2 + a^2\sin^2\alpha = a^2 + d^2 + 2ad\cos\alpha.$$

$$|d + rz|^2 = (ar\cos\alpha + d)^2 + a^2r^2\sin^2\alpha = a^2r^2 + d^2 + 2adr\cos\alpha.$$

Inequality (3.4) is thus equivalent to the following inequality:

$$(d+2)^2(a^2r^2 + d^2 + 2adr\cos\alpha) \ge (d+2r)^2(a^2 + d^2 + 2ad\cos\alpha). \tag{3.5}$$

After performing the multiplications, canceling the same terms, and simplifying by $(1-r)$, this amounts to verifying that

$$4d^3 + 4d^2(1+r) + 8adr\cos\alpha \ge a^2d^2(1+r) + 2ad^3\cos\alpha + 4a^2dr. \tag{3.6}$$

Observe that

$$4d^2(1+r) \ge a^2d^2(1+r).$$

It remains to show that (after simplifying by $2d$):

$$2d^2 + 4ar\cos\alpha \ge ad^2\cos\alpha + 2a^2r. \tag{3.7}$$

This last inequality is equivalent to

$$2(d^2 - a^2r) \ge a(d^2 - 4r)\cos\alpha. \tag{3.8}$$

Inequality (3.8) is clearly satisfied when $\cos\alpha < 0$, so assume now that $\cos\alpha \ge 0$. Obviously $2 \ge a\cos\alpha$, and from $a^2 \le 4$, we also get

$$d^2 - a^2r \ge d^2 - 4r.$$

Putting these two inequalities together (taking the product) gives inequality (3.8), hence inequality (3.2) is proved.

To see that (3.2) is tight, take $p_i = (-1, 0)$, and $p_j = (d+1, 0)$, i.e., all six points $p_i, p_j, o_i, o_j, q_i, q_j$ are on the same line. The proof of Lemma 3.2 is now complete. ■

Lemma 3.3. *Let $p_1, \ldots, p_n$ be n points, where $p_i \in \Omega_i$, such that for any $i \ne j$, $|p_ip_j| \ge d$, for some $d > 0$. Then there exist n points, $q_1, \ldots, q_n$, such that $q_i \in \omega_i$, and for any $i \ne j$, $|q_iq_j| \ge \frac{\delta+2r}{\delta+2} \cdot d$.*

Proof. Let q_i be defined as in Lemma 3.2. It suffices to show that

$$\frac{|q_iq_j|}{|p_ip_j|} \ge \frac{\delta+2r}{\delta+2}.$$

By Lemma 3.2,

$$\frac{|q_iq_j|}{|p_ip_j|} \ge \frac{|o_io_j| + 2r}{|o_io_j| + 2}.$$

Since $|o_io_j| \ge \delta$, we obviously have

$$\frac{|o_io_j| + 2r}{|o_io_j| + 2} \ge \frac{\delta+2r}{\delta+2}.$$

By combining the two inequalities the lemma follows. ■

For $\delta \geq 2$, and $0 < r \leq 1$, let

$$c_1(\delta, r) = \frac{\delta + 2r}{\delta + 2}, \quad c_2(\delta, r) = \frac{\sqrt{\delta^2 - 4r^2}}{\delta}.$$

Observe that $c_1(\delta, r) \leq 1$, and $c_2(\delta, r) \leq 1$. We will show that Step 1 and Step 2 can be implemented as to achieve approximation ratios $c_1(\delta, r)$ and $c_2(\delta, r)$, respectively. The resulting overall approximation ratio is then

$$c(\delta, r) = c_1(\delta, r) \cdot c_2(\delta, r),$$

and it remains to choose $r = r(\delta)$ over the whole range $\delta \geq 2$, so as to maximize $c(\delta, r)$.

Selecting $r(\delta)$**.** For a fixed $\delta \geq 2$, let

$$f(r) = c(\delta, r) = c_1(\delta, r) \cdot c_2(\delta, r) = \frac{\delta + 2r}{\delta + 2} \cdot \frac{\sqrt{\delta^2 - 4r^2}}{\delta}.$$

Note that $r \leq 1 \leq \frac{\delta}{2}$, hence $f(r)$ is well defined.

Consider first the case $2 \leq \delta < 4$. Assume further that $r < 1$, so that $\sqrt{\delta^2 - 4r^2}$ and $f(r)$ are strictly positive. The derivative of $f(r)$ is

$$f'(r) = \frac{2(\delta + 2r)(\delta - 4r)}{\delta(\delta + 2)\sqrt{\delta^2 - 4r^2}}. \tag{3.9}$$

The function $f(r)$ is maximized by setting $f'(r)$ to zero, which yields $r = \frac{\delta}{4}$, (note that $r < 1$), and correspondingly,

$$c\left(\delta, \frac{\delta}{4}\right) = c_1\left(\delta, \frac{\delta}{4}\right) \cdot c_2\left(\delta, \frac{\delta}{4}\right) = \frac{3\delta}{2} \cdot \frac{1}{\delta + 2} \cdot \sqrt{\frac{3}{4}} = \frac{3\sqrt{3}}{4} \cdot \frac{\delta}{\delta + 2}.$$

Observe that $c(\delta, \frac{\delta}{4}) \geq c(2, \frac{1}{2}) = 0.6495\ldots$, in our interval $2 \leq \delta < 4$.

Consider now the case $\delta \geq 4$, and assume further that $r \leq 1$. Since $\delta \geq 4 > 2$, the expression of the derivative $f'(r)$ in equation (3.9) is still valid. We have $f'(r) > 0$, hence $f(r)$ is an increasing function, so

$$c(\delta, r) = f(r) \leq f(1) = \frac{\sqrt{\delta^2 - 4}}{\delta}.$$

Thus for $\delta \geq 4$, we set $r = 1$. To summarize, we set

$$r = r(\delta) = \begin{cases} \frac{\delta}{4} & \text{if } 2 \leq \delta \leq 4, \\ 1 & \text{if } \delta \geq 4. \end{cases} \tag{3.10}$$

Note that $r(\delta)$ is a continuous function over the entire range $\delta \geq 2$. The resulting overall approximation ratio of the algorithm, denoted by $c = c(\delta)$, is at least

$$c(\delta) = \begin{cases} \frac{3\sqrt{3}}{4} \cdot \frac{\delta}{\delta+2} & \text{if } 2 \leq \delta \leq 4, \\ \frac{\sqrt{\delta^2-4}}{\delta} & \text{if } \delta \geq 4. \end{cases} \tag{3.11}$$

Define also for future reference the approximation ratios achieved in Step 1 and Step 2 of the algorithm, based on our previous choice of r, depending on δ.

$$c_1 = c_1(\delta) = \begin{cases} \frac{3}{2} \cdot \frac{\delta}{\delta+2} & \text{if } 2 \leq \delta \leq 4, \\ 1 & \text{if } \delta \geq 4. \end{cases} \tag{3.12}$$

$$c_2 = c_2(\delta) = \begin{cases} \frac{\sqrt{3}}{2} & \text{if } 2 \le \delta \le 4, \\ \frac{\sqrt{\delta^2-4}}{\delta} & \text{if } \delta \ge 4. \end{cases} \tag{3.13}$$

In particular, for $\delta = 2$, we have

$$r = \frac{1}{2}, \quad c_1 = \frac{3}{4}, \quad c_2 = \frac{\sqrt{3}}{2},$$

hence the overall ratio for Step 1 and Step 2 is $c_1 c_2 = \frac{3\sqrt{3}}{8}$.

To implement Step 2, we are lead to the following linear program, with the constraints expressed symbolically at this point. LP1 maximizes the minimum projection on the set of lines connecting the centers of the disks; that is, for each pair of disks, the length of the projection of the segment connecting the corresponding two points on the line connecting the two disk centers.

$$\begin{array}{ll} \text{maximize} & z \\ \text{subject to} & \begin{cases} p_i \in \omega_i, & 1 \le i \le n \\ \text{proj}_{\alpha_{ij}}(p_i, p_j) \ge z, & 1 \le i < j \le n \end{cases} \end{array} \tag{LP1}$$

Approximating the small disks by regular polygons. Let $\lambda > 0$ be small. Recall that $r = r(\delta)$ is a fixed precomputed value. Select k large enough so that the apothem of the regular k-gon inscribed in a circle of radius r is at least $r(1-\lambda)$. Recall that the apothem length a is given by the formula: $a = r\cos\frac{\pi}{k}$, so we need to choose k so that

$$\cos\frac{\pi}{k} \ge 1 - \lambda. \tag{3.14}$$

The symbolic constraint $p_i \in \omega_i$ is replaced by the k linear constraints defining the sides of the regular polygon (the polygon is the intersection of k half-planes). Let $\varepsilon > 0$ be small. By setting $\lambda = \lambda(\varepsilon)$ sufficiently small, we can ensure that the approximation ratio remains at least $(1-\varepsilon)\frac{3\sqrt{3}}{8}$, say at least 0.649. Let now

$$c_3(\delta, r) = \frac{\delta + 2r(1-\lambda)}{\delta + 2r}. \tag{3.15}$$

Replacing the small disks of radius r by regular polygons with k sides incurs only a slight loss in the approximation ratio for k sufficiently large, since the disks of radii a are contained in the regular polygons with k sides, and a is close to r. Analogous to inequality (3.2) in Lemma 3.2, the setting in (3.15) is justified, and the overall approximation ratio of the algorithm is at least $c_3(\delta, r) \cdot c(\delta, r)$. Recall the setting of $r(\delta)$ given by (3.10). For $2 \le \delta \le 4$, we have

$$c_3\left(\delta, \frac{\delta}{4}\right) = \frac{\delta + 2\frac{\delta}{4}(1-\lambda)}{\delta + 2\frac{\delta}{4}} = 1 - \frac{\lambda}{3}.$$

For $\delta \ge 4$, we have

$$c_3(\delta, 1) = \frac{\delta + 2(1-\lambda)}{\delta + 2} = 1 - \frac{2\lambda}{\delta + 2} \ge 1 - \frac{\lambda}{3}.$$

Consequently, to ensure that the approximation ratio of the algorithm is at least $(1-\varepsilon)\cdot c(\delta)$ over the entire range $\delta \geq 2$, let $\lambda = 3\varepsilon$, and choose k such that (recall (3.14)):

$$\cos\frac{\pi}{k} \geq 1 - 3\varepsilon.$$

For instance, setting $\varepsilon = \frac{7}{10000}$, and $k = 50$ satisfies the above inequality and ensures that the approximation ratio remains at least $(1-\varepsilon)\frac{3\sqrt{3}}{8} \geq 0.649$.

Writing the linear constraints. Implement each symbolic constraint $\mathrm{proj}_{\alpha_{ij}}(p_i, p_j) \geq z$ as follows: Let $o_i = (\xi_i, \eta_i)$ be coordinates of o_i, for $i = 1, \ldots, n$ (part of the input). For simplicity, assume that the disk centers are non-decreasing order of their x-coordinates: $\xi_1 \leq \xi_2 \leq \ldots \leq \xi_n$. Consider a pair i, j, where $i < j$. Recall that $\alpha_{ij} \in (-\pi/2, \pi/2)$ is the angle of the line determined by o_i and o_j. We have

$$\cos\alpha_{ij} = \frac{\xi_j - \xi_i}{|o_i o_j|}, \quad \sin\alpha_{ij} = \frac{\eta_j - \eta_i}{|o_i o_j|}. \tag{3.16}$$

Let $\overline{a_{ij}} = (\cos\alpha_{ij}, \sin\alpha_{ij})$, so that $|\overline{a_{ij}}| = 1$. Let $\overline{s_{ij}} = (x_j - x_i, y_j - y_i)$. According to (3.1),

$$\mathrm{proj}_{\alpha_{ij}}(p_i, p_j) = \frac{\langle \overline{a_{ij}} \cdot \overline{s_{ij}} \rangle}{|\overline{a_{ij}}|} = \langle \overline{a_{ij}} \cdot \overline{s_{ij}} \rangle = (x_j - x_i)\cos\alpha_{ij} + (y_j - y_i)\sin\alpha_{ij}.$$

Consequently, for each pair i, j, where $i < j$, generate the constraint:

$$(x_j - x_i)\cos\alpha_{ij} + (y_j - y_i)\sin\alpha_{ij} \geq z;$$

where $\cos\alpha_{ij}$ and $\sin\alpha_{ij}$ are as in (3.16).

Establishing the approximation ratio.

Lemma 3.4. *Let $p_1, \ldots, p_n$ be n points, where $p_i \in \omega_i$, such that for any $i \neq j$, $|p_i p_j| \geq d$, for some $d > 0$. Then for any $i \neq j$, $\mathrm{proj}_{\alpha_{ij}}(p_i, p_j) \geq c_2 \cdot d$.*

Proof. Observe that the line determined by the points p_i and p_j intersects both disks ω_i and ω_j. The claimed inequality is now immediate from Lemma 3.1. ■

Lemma 3.5. *Let $p_1, \ldots, p_n$ be n points, where $p_i \in \omega_i$, such that for any $i \neq j$, $\mathrm{proj}_{\alpha_{ij}}(p_i, p_j) \geq d$, for some $d > 0$. Then, for any $i \neq j$, $|p_i p_j| \geq d$.*

Proof. Obviously, $|p_i p_j| \geq \mathrm{proj}_{\alpha_{ij}}(p_i, p_j) \geq d$, as required. ■

Lemma 3.6. *The ratio of the approximation algorithm* **A2** *is at least $(1-\varepsilon)\frac{3\sqrt{3}}{8}$, for any given $\varepsilon > 0$. ($\frac{3\sqrt{3}}{8} = 0.6495\ldots$) Moreover, if $\delta \geq 2$ is the minimum distance among the unit disk centers, the approximation ratio is at least $(1-\varepsilon)\cdot c(\delta) \geq (1-\varepsilon)\frac{3\sqrt{3}}{8}$, where $c(\delta)$ is given by* (3.11)*.*

Proof. Let $p_1, \ldots, p_n$ be n points, where $p_i \in \Omega_i$, such that for any $i \neq j$, $|p_i p_j| \geq d$, for some $d > 0$. In other words, assume that $\mathrm{OPT} \geq d$. By Lemma 3.3, there exist n points, $q_1, \ldots, q_n$, such that $q_i \in \omega_i$, and for any $i \neq j$, $|q_i q_j| \geq c_1 \cdot d$. (This inequality is trivial for $\delta \geq 4$, since we set $r = 1$, and $c_1 = 1$ in that case; refer to (3.12).) By Lemma 3.4, for any $i \neq j$, $\mathrm{proj}_{\alpha_{ij}}(q_i, q_j) \geq c_2 \cdot c_1 \cdot d = c(\delta) \cdot d$. Recall that the linear program (LP1) finds a point set $\{p_i = (x_i, y_i), i = 1, \ldots, n\}$, for which the minimum projection is maximized. However, the feasible regions for each point are the slightly smaller inscribed regular polygons rather than the small disks. By Lemma 3.5, and the preceding discussion, the computed point set satisfies that, for any $i \neq j$, $|p_i p_j| \geq (1-\varepsilon) \cdot c(\delta) \cdot d$. Hence the approximation algorithm has ratio at least $(1-\varepsilon) \cdot c(\delta) \geq (1-\varepsilon)\frac{3\sqrt{3}}{8}$, as claimed. ■

Reducing the number of constraints to $O(n)$. Recall that OPT $\leq \delta + 2$. So there is no need to write any constraints for pairs of disks at distance $\delta + 4$ or more, since the distance between the corresponding points is at least $\delta + 2$. An easy packing argument shows that the number of pairs of disks at distance at most $\delta + 4$ is only $O(n)$.

Solving the LP. The constraints of the LP involve irrational numbers, and hence it cannot be claimed that the original LP is solvable in polynomial time. However, it is enough to solve the LP up to some precision. For this, it is enough to approximate the numbers involved in the constraints up to some precision, which is polynomial in the error of the output. There are bounds on how many bits of precision are needed in the constraints to obtain a bound on the precision of the solution, and they are polynomially related [9]. Consequently, since we are dealing with ε-approximation anyway, we can encode each coefficient into a rational number with $(1/\varepsilon)^{O(1)}$ bits. Then, by our choice of ε, each coefficient has a constant number of bits. Thus the LP algorithm runs in polynomial time; e.g., $O(n^4)$ or $O(n^{3.5})$ using interior point methods.

References

[1] C. Baur and S.P. Fekete: Approximation of geometric dispersion problems, *Algorithmica*, **30** (2001), 451–470.

[2] M. Benkert, J. Gudmundsson, C. Knauer, R. van Oostrum, and A. Wolff: A polynomial-time approximation algorithm for a geometric dispersion problem, *International Journal of Computational Geometry and Applications*, **19(3)** (2009), 267–288.

[3] S. Cabello: Approximation algorithms for spreading points, *Journal of Algorithms*, **62** (2007), 49–73.

[4] E.D. Demaine, M. Hajiaghayi, H. Mahini, A.S. Sayedi-Roshkhar, S. Oveisgharan, and M. Zadimoghaddam: Minimizing movement, in *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007, pp. 258–267.

[5] H. Edelsbrunner: *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[6] J. Fiala, J. Kratochvíl, and A. Proskurowski: Systems of distant representatives, *Discrete Applied Mathematics*, **145** (2005), 306–316.

[7] M. Formann and F. Wagner: A packing problem with applications to lettering of maps, in *Proceedings of the 7th Annual Symposium on Computational Geometry*, 1991, pp. 281–288.

[8] M. Jiang, S. Bereg, Z. Qin, and B. Zhu: New bounds on map labeling with circular labels, in *Proceedings of the 15th Annual International Symposium on Algorithms and Computation*, 2004, pp. 606–617.

[9] A. Schrijver: *Theory of Linear and Integer Programming*, John Wiley & Sons, New York, 1986.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 311-322
www.stacs-conf.org

LONG NON-CROSSING CONFIGURATIONS IN THE PLANE

ADRIAN DUMITRESCU[1] AND CSABA D. TÓTH[2]

[1] Department of Computer Science, University of Wisconsin–Milwaukee, WI 53201-0784, USA
E-mail address: ad@cs.uwm.edu

[2] Department of Mathematics and Statistics, University of Calgary, AB, Canada
E-mail address: cdtoth@ucalgary.ca

Abstract. We revisit several maximization problems for geometric networks design under the non-crossing constraint, first studied by Alon, Rajagopalan and Suri (ACM Symposium on Computational Geometry, 1993). Given a set of n points in the plane in general position (no three points collinear), compute a longest non-crossing configuration composed of straight line segments that is: (a) a matching (b) a Hamiltonian path (c) a spanning tree. Here we obtain new results for (b) and (c), as well as for the Hamiltonian cycle problem:

(i) For the longest non-crossing Hamiltonian path problem, we give an approximation algorithm with ratio $\frac{2}{\pi+1} \approx 0.4829$. The previous best ratio, due to Alon et al., was $1/\pi \approx 0.3183$. Moreover, the ratio of our algorithm is close to $2/\pi$ on a relatively broad class of instances: for point sets whose perimeter (or diameter) is much shorter than the maximum length matching. The algorithm runs in $O(n^{7/3} \log n)$ time.

(ii) For the longest non-crossing spanning tree problem, we give an approximation algorithm with ratio 0.502 which runs in $O(n \log n)$ time. The previous ratio, 1/2, due to Alon et al., was achieved by a quadratic time algorithm. Along the way, we first re-derive the result of Alon et al. with a faster $O(n \log n)$-time algorithm and a very simple analysis.

(iii) For the longest non-crossing Hamiltonian cycle problem, we give an approximation algorithm whose ratio is close to $2/\pi$ on a relatively broad class of instances: for point sets with the product $\langle$ diameter $\times$ convex hull size $\rangle$ much smaller than the maximum length matching. The algorithm runs in $O(n^{7/3} \log n)$ time. No previous approximation results were known for this problem.

1. Introduction

Self-crossing in planar configurations is typically an undesirable attribute. Many structures studied in computational geometry, in particular those involving a minimization condition, have the non-crossing attribute for free, for instance minimum spanning trees, minimum length matchings, Voronoi diagrams, etc. The non-crossing property usually follows

1998 ACM Subject Classification: F.2.2 Geometrical problems and computations.

Key words and phrases: Longest non-crossing Hamiltonian path, longest non-crossing Hamiltonian cycle, longest non-crossing spanning tree, approximation algorithm.

Adrian Dumitrescu was supported in part by NSF CAREER grant CCF-0444188. Part of the research by this author was done at Ecole Polytechnique Fédérale de Lausanne. Csaba D. Tóth was supported in part by NSERC grant RGPIN 35586. Part of the research by this author was done at Tufts University.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2465

from the triangle inequality. Alon et al. [3] have considered the problems of computing (i) the longest non-crossing matching, (ii) the longest non-crossing Hamiltonian path and (iii) the longest non-crossing spanning tree, given n points in the plane. Although they were unable to prove it, they suspected that all these problems are NP-hard. The survey articles by Eppstein [8, pp. 439] and Mitchell [14, pp. 680] list these as open problems in the area of geometric network optimization. The problem of approximating the longest non-crossing Hamiltonian cycle is also of interest and wide open [4, pp. 338].

Without the non-crossing condition explicitly enforced, the problem of minimizing or maximizing the length of a spanning tree, Hamiltonian cycle or path, perfect matching, triangulation, etc. has a rich history. However if such structures are required to be non-crossing much less is known, in particular for the maximization variants. While for minimization problems, the non-crossing property comes usually for free via the triangle inequality, in contrast, for maximization problems, the non-crossing property conflicts directly with the length maximizing objective. This is another reason why these problems are interesting to study.

Related work. The existence of non-crossing Hamiltonian paths and cycles in geometric graphs has been studied in [2, 5]. Various Ramsey-type results for non-crossing spanning trees, paths and cycles have been obtained in [11] and [12]. The Euclidean MAX TSP, the problem of computing a longest straight-line tour of a set of points, has been proven NP-hard in dimensions three or higher [9], while its complexity in the Euclidean plane remains open [14]. In contrast, the shortest non-crossing matching and the shortest non-crossing spanning tree are both computable in polynomial time [8, 14], as they coincide with the shortest matching and the shortest spanning tree respectively.

Definitions and notations. A set S of points in the plane is said to be in *general position* if no three points are collinear. General position will be assumed throughout this paper. Given a set of n points in the plane, the results of Alon al. are as follows: (i) A non-crossing matching whose total length is at least $2/\pi$ of the longest (possibly crossing) matching can be computed in $O(n^{7/3}\log n)$ time. (ii) A non-crossing Hamiltonian path whose total length is at least $1/\pi$ of the longest (possibly crossing) Hamiltonian path can be computed in $O(n^{7/3}\log n)$ time. (iii) A non-crossing spanning tree whose total length is at least $n/(2n-2) \geq 1/2$ of the longest (possibly crossing) spanning tree can be computed in $O(n^2)$ time. The running times have been adjusted to reflect the current best upper bound of $O(n^{4/3})$ on the number of halving lines as established by Dey [6].

A *geometric graph* G is a pair (V, E) where V is a finite set of points in general position in the plane, and E is a set set of segments (edges) connecting points in V. The *length* of G, denoted $L(G)$, is the sum of the Euclidean lengths of all edges in G. The graph G is said to be *non-crossing* if its edges have pairwise disjoint interiors (collinear triples of points are forbidden in order to avoid overlapping collinear edges).

For a point set S, let $\text{conv}(S)$ be the convex hull of S, and let $P = P(S)$ denote the perimeter of $\text{conv}(S)$. Denote by $D = D(S)$ the diameter of S and write $n = |S|$. Let M_{OPT} be a longest (possibly crossing) matching of S, and let M^*_{OPT} be a longest non-crossing matching of S; observe that for odd n, M_{OPT} is a nearly perfect matching, with $(n-1)/2$ edges. Let H_{OPT} be a longest (possibly crossing) Hamiltonian path of S, and let H^*_{OPT} be a longest non-crossing Hamiltonian path of S. Let T_{OPT} be a longest (possibly crossing) spanning tree of S, and let T^*_{OPT} be a longest non-crossing spanning tree of S. Finally, let Q_{OPT} be a longest (possibly crossing) Hamiltonian cycle of S, and let Q^*_{OPT}

be a longest non-crossing Hamiltonian cycle of S. The following inequalities are obvious: $L(M_{\mathrm{OPT}}) \leq L(H_{\mathrm{OPT}}) \leq L(T_{\mathrm{OPT}})$.

Given a set S of n points in the plane, a line ℓ going through two points of S is called a *halving line* if there are $\lfloor (n-2)/2 \rfloor$ points on one side and $\lceil (n-2)/2 \rceil$ points on the other side [13]. A *bisecting line* ℓ of S is any line that partitions the point set evenly, i. e., neither of the two open halfplanes defined by ℓ contains more than $n/2$ points of S [7]. Observe that any halving line of S is also a bisecting line of S. Any bisecting line of S yields (perhaps non-uniquely) a bipartition $S = R \cup B$, with $R \cap B = \emptyset$, $||R| - |B|| \leq 1$, with R contained in one of the closed halfplanes determined by ℓ, and B contained in the other. We call $S = R \cup B$ a linearly separable bipartition, or balanced partition of S. Observe that for any non-zero direction vector $\vec{v}$, there is a bisecting line orthogonal to $\vec{v}$, see [7, Lemma 4.4]. Two bisecting lines are called *equivalent* if they can yield the same balanced partition of S. It is well known that the number of non-equivalent bisecting lines of a set is of the same order as the number of halving lines of the set, and any balanced bipartition can be obtained from a halving line [7, pp. 67].

Our results are summarized in the following three theorems[1].

Theorem 1.1. (i) *For the longest non-crossing Hamiltonian path problem, there is an approximation algorithm with ratio* $\frac{2}{\pi+1} \approx 0.4829$ *that runs in* $O(n^{7/3} \log n)$ *time.*
(ii) *Given a set of n points in the plane, one can compute a non-crossing Hamiltonian path H in $O(n^{7/3} \log n)$ time such that $L(H) \geq \frac{2}{\pi} L(H_{OPT}) - \frac{P}{\pi}$. In particular, if the point set satisfies the condition $\frac{P}{\pi} \leq \delta L(H_{OPT})$ for some small $\delta > 0$, then $L(H) \geq (\frac{2}{\pi} - \delta) L(H_{OPT})$.*
(iii) *Alternatively, one can compute a non-crossing Hamiltonian path H in $O(n \log n / \sqrt{\varepsilon})$ time, such that $L(H) \geq (1-\varepsilon)\frac{2}{\pi} L(H_{OPT}) - \frac{P}{\pi}$.*

Theorem 1.2. *For the longest non-crossing spanning tree problem for a given set of n points in the plane, there is an approximation algorithm with ratio* 0.502 *and* $O(n \log n)$ *running time. More precisely, the algorithm computes a non-crossing spanning tree T such that $L(T) \geq 0.502 \cdot L(T_{OPT})$.*

Theorem 1.3. *Given a set S of n points in the plane, with $|\mathrm{conv}(S)| = h$:*
(i) *One can compute a non-crossing Hamiltonian cycle Q in $O(n^{7/3} \log n)$ time such that $L(Q) \geq \frac{2}{\pi} L(Q_{OPT}) - (2h-1)\frac{P}{\pi}$. In particular, if the point set satisfies the condition $(2h-1)\frac{P}{\pi} \leq \delta L(Q_{OPT})$ for some small $\delta > 0$, then $L(Q) \geq \left(\frac{2}{\pi} - \delta\right) L(Q_{OPT})$.*
(ii) *Alternatively, one can compute a non-crossing Hamiltonian cycle Q in $O(n^3 \log n)$ time such that $L(Q) \geq \frac{2}{\pi} L(Q_{OPT}) - (h+2)\frac{P}{\pi}$.*
(iii) *Alternatively, one can compute a non-crossing Hamiltonian cycle Q in $O(n \log n / \sqrt{\varepsilon})$ time, such that $L(Q) \geq (1-\varepsilon)\frac{2}{\pi} L(Q_{OPT}) - (2h-1)\frac{P}{\pi}$.*

2. The Hamiltonian path

In this section we prove Theorem 1.1. Let $S = \{p_1, \ldots, p_n\}$. We follow an approach similar to that of Alon et al. using projections and an averaging argument, in conjunction with a result on bipartite embeddings of spanning paths in the plane. Abellanas et al. [1, Theorem 3.1] showed that every linearly separable bipartition $S = R \cup B$ with $||R| - |B|| \leq 1$, admits an alternating non-crossing spanning path such that the edges cross any separating

[1] Due to space limitations, some proofs are omitted.

line ℓ at points ordered monotonically along ℓ. Such a Hamiltonian path can be computed in $O(n \log n)$ time. Their algorithm computes the same Hamiltonian path for any two equivalent halving lines, that is, the alternating path depends on the bipartition only rather than the separating line.

We now recall the algorithm of Abellanas et al. [1]; see Fig. 4 for an example. Let $S = R \cup B$ with $||R| - |B|| \leq 1$ be the red-blue bipartition given by a vertical line ℓ: R on the left, B on the right. Their algorithm constructs an alternating path A in the following way: Let rb be the top red-blue edge of the convex hull $\text{conv}(S)$, called the *top bridge*. If $|R| > |B|$, set $A := \{r\}$, if $|R| < |B|$, set $A := \{b\}$, else set A to $\{r\}$ or $\{b\}$ arbitrarily. At every step, recompute the top bridge rb of $S \setminus A$, and add r to A if the last point in A was blue, or add b to A if the last point in A was red. As pointed out by the authors, the resulting path A is non-crossing because A is disjoint from the convex hull of $S \setminus A$ at each step.

We improve the lower bound of Alon et al. by computing the longest Hamiltonian path corresponding to a bipartition and a Hamiltonian path of length at least the perimeter of the convex hull, and returning the longest of the two.

Lemma 2.1. *For a point set S, $|S| = n \geq 31$, a non-crossing Hamiltonian path of length at least $P(S)$ can be computed in $O(n \log n)$ time. The bound on the length is best possible.*

Consider a geometric graph $G = (V, E)$, and a point $q \notin V$, so that $V \cup \{q\}$ is in general position. We say that *q sees a vertex $v \in V$* if the segment qv does not intersect any edge of G. Similarly, we say that *q sees an edge $e \in E$*, if the triangle formed by v and e does not intersect any other edge of G. We make use of the fact that if n is even then the two endpoints of an alternating path are on opposite sides of the separating line ℓ. If n is odd, we first construct an alternating path for a specific subset of $n-1$ points, and then augment it to a Hamiltonian path on all n points using the following lemma.

Lemma 2.2. *Let $S = R \cup B$ with $||R| - |B|| \leq 1$, be a linearly separable bipartition given by line ℓ. Let $q \in S$, and A' be a non-crossing alternating path on $S \setminus \{q\}$ such that its (consecutive) edges cross ℓ at points ordered monotonically along ℓ. Then q sees one edge of A' and consequently, A' can be extended to a Hamiltonian path A on S, with $L(A') < L(A)$. The path A can be computed in $O(n)$ time, given A'.*

Fix a Cartesian coordinate system Γ. Let k be the number of halving lines of S, denote the angles they make with the x-axis of Γ by $0 \leq \alpha_1 < \ldots \alpha_k < \pi$. By relabeling the points assume that the optimal path is $H_{\text{OPT}} = p_1, p_2, \ldots, p_n$. For two points $p_i, p_j \in S$, let β_{ij} be the angle in $[0, \pi)$ formed by the line through $p_i p_j$ and the x-axis. If n is odd, then a bisecting line of direction α (for any α) must be incident to at least one point of S, and denote an arbitrary such point by q_α.

Algorithm **A1**:
Step 1. Compute a non-crossing Hamiltonian path H_1 of length at least $P(S)$, by Lemma 2.1.
Step 2. If n is even, then for all non-equivalent bisections of S (i.e., for all balanced bipartitions of S), compute a non-crossing alternating path using the algorithm of Abellanas et al. [1], and let the longest such path be H_2. If n is odd, then for all non-equivalent bisections of S, compute a non-crossing alternating path of the even point set $S \setminus \{q_\alpha\}$ using the algorithm of [1] and let the longest such path be H_2'. Augment H_2' with vertex q_α by Lemma 2.2 to a Hamiltonian path H_2.
Step 3. Output the longest of the two paths H_1 and H_2.

By Lemma 2.1, the running time of Step 1 is $O(n \log n)$. Since the number of halving lines of an n-element point set is $O(n^{4/3})$ and all can be generated within this time [6], the running time of Step 2 is $O(n^{7/3} \log n)$, consequently the total running time of **A1** is also $O(n^{7/3} \log n)$.

We proceed with the analysis of the approximation ratio. For simplicity, we assume that n is even. The case of n odd is slightly different. For each $\alpha \in [0, \pi)$, let Γ_α be a (rotated) coordinate system, obtained from Γ via a counterclockwise rotation by α, and with the y-axis dividing evenly the point set S. Let x_i be the x-coordinate of point p_i with respect to Γ_α. For a given α, let H_α be a non-crossing alternating path with respect to a balanced bipartition induced by the y-axis of Γ_α, as computed by the algorithm. There are $O(1)$ balanced bipartitions given by any halving line of S. Recall that H_α does not depend continuously on α; it depends only on the discrete bipartition. However, the coordinates of the points depend continuously on α. Assume that $H_\alpha = p_{\sigma(1)}, p_{\sigma(2)}, \ldots, p_{\sigma(n)}$, where σ is a permutation of $[n]$; here σ depends on the bipartition (hence also on α). Let W_α denote the *width* of S in direction α, that is, the width of the smallest parallel strip of direction α that contains S. By projecting on the x-axis of Γ_α, we get

$$\begin{aligned}
L(H_\alpha) &\geq |x_{\sigma(1)}| + 2|x_{\sigma(2)}| + \ldots + 2|x_{\sigma(n-1)}| + |x_{\sigma(n)}| = 2\sum_{i=1}^{n} |x_i| - |x_{\sigma(1)}| - |x_{\sigma(n)}| \\
&= \sum_{j=1}^{n-1} (|x_j| + |x_{j+1}|) + |x_1| + |x_n| - |x_{\sigma(1)}| - |x_{\sigma(n)}| \geq \sum_{j=1}^{n-1} (|x_j| + |x_{j+1}|) - W_\alpha \\
&\geq \sum_{j=1}^{n-1} |p_j p_{j+1}| |\cos(\beta_{jj+1} - \alpha)| - W_\alpha \qquad (2.1)
\end{aligned}$$

In the 2nd line of the above chain of inequalities, we use the fact that $p_{\sigma(1)}$ and $p_{\sigma(n)}$ lie on opposite sides of ℓ, since n is even, hence $|x_{\sigma(1)}| + |x_{\sigma(n)}| \leq |p_{\sigma(1)} p_{\sigma(n)}| \leq W_\alpha$, In the 3rd line, we make use of the following inequality: for any two points $p_i, p_j \in S$, $|p_i p_j| |\cos(\beta_{ij} - \alpha)| \leq |x_i| + |x_j|$, with equality if and only if the two points lie on opposite sides of the y-axis of Γ_α. Recall: for even n, H_2 is the longest of the $O(k)$ Hamiltonian non-crossing paths H_{α_i} over all $O(k)$ balanced bipartitions of S. (A given angle α_i yields $O(1)$ balanced partitions, and corresponding alternating paths denoted here H_{α_i}.) We thus have for each $\alpha \in [0, \pi)$:

$$L(H_2) \geq \sum_{j=1}^{n-1} |p_j p_{j+1}| |\cos(\beta_{jj+1} - \alpha)| - W_\alpha.$$

Note that

$$\int_0^\pi |\cos(\beta_{jj+1} - \alpha)| \,\mathrm{d}\alpha = \int_0^\pi |\cos\alpha| \,\mathrm{d}\alpha = 2,$$

and according to Cauchy's surface area formula, we have $\int_0^\pi W_\alpha \,\mathrm{d}\alpha = P(S)$. By integrating both sides of the previous inequality over the α-interval $[0, \pi]$, we obtain

$$\pi L(H_2) \geq 2\sum_{j=1}^{n-1} |p_j p_{j+1}| - P(S) = 2L(H_{\mathrm{OPT}}) - P(S),$$

$$L(H_2) \geq \frac{2}{\pi} L(H_{\mathrm{OPT}}) - \frac{P(S)}{\pi}. \qquad (2.2)$$

We now improve the old approximation ratio of $\frac{1}{\pi} \approx 0.3183$ to $\frac{2}{\pi+1} \approx 0.4829$, by balancing the lengths of the two paths computed in Step 1 and Step 2. Set $c = \frac{\pi+1}{2}$.

Case 1: $L(H_{\text{OPT}}) \leq cP(S)$. By considering the path computed in Step 1, we get a ratio of at least

$$\frac{L(H_1)}{L(H_{\text{OPT}})} \geq \frac{P(S)}{L(H_{\text{OPT}})} \geq \frac{P(S)}{cP(S)} = \frac{2}{\pi+1}.$$

Case 2: $L(H_{\text{OPT}}) \geq cP(S)$. By considering the path computed in Step 2 (inequality (2.2)), we get a ratio of at least

$$\frac{L(H_2)}{L(H_{\text{OPT}})} \geq \frac{\frac{2}{\pi}L(H_{\text{OPT}}) - \frac{1}{\pi}P(S)}{L(H_{\text{OPT}})} \geq \frac{2}{\pi} - \frac{1}{c\pi} = \frac{2}{\pi}\left(1 - \frac{1}{\pi+1}\right) = \frac{2}{\pi+1}.$$

Observe that if the point set satisfies the condition $\frac{P(S)}{\pi} \leq \delta L(H_{\text{OPT}})$, then by (2.2), we have

$$L(H) \geq \frac{2}{\pi}L(H_{\text{OPT}}) - \delta L(H_{\text{OPT}}) = \left(\frac{2}{\pi} - \delta\right) L(H_{\text{OPT}}).$$

This concludes the proofs of parts (i) and (ii) of Theorem 1.1.

(iii) With the same approach as in [3], a Hamiltonian path of length at least $(1-\varepsilon)\frac{2}{\pi}L(H_{\text{OPT}}) - \frac{P(S)}{\pi}$ can be found by considering only $b/\sqrt{\varepsilon}$ angles $\theta_i = \frac{i\pi\sqrt{\varepsilon}}{b}$, for $i = 0, 1, \ldots, \lfloor b/\sqrt{\varepsilon} \rfloor$, where b is a suitable absolute constant. The resulting running time is $O(n \log n/\sqrt{\varepsilon})$. This concludes the proof of Theorem 1.1.

3. The spanning tree

In this section we prove Theorem 1.2. Let $S = \{p_1, \ldots, p_n\}$, where $p_i = (x_i, y_i)$. Given a point $p \in S$, the *star centered at* p, denoted S_p, is the spanning tree on S whose edges join p to all the other points. Since S is in general position, S_p is non-crossing for any $p \in S$. An *extended star centered at* p is a spanning tree of S consisting of paths of length 1 or 2 (edges) connecting p to all the other points. See Fig. 1. While the star centered at a point is unique, there may be many extended stars centered at the same point, and some of them may be self-crossing. In particular S_p is also an extended star.

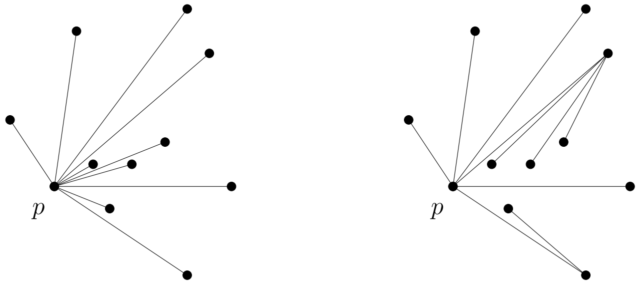


Figure 1: A star (left) and a non-crossing extended star (right) on a same point set, both centered at the same point p.

The algorithm of Alon et al. computes the n stars centered at each of the points, and then outputs the longest one. The algorithm takes quadratic time, and the analysis shows a ratio of $\frac{n}{2n-2}$ (which tends to 1/2 in the limit). Their algorithm works in any metric space.

As pointed out by Alon et al., the ratio 1/2 is best possible (in the limit) for this specific algorithm. We first re-establish the 1/2 approximation ratio using a faster algorithm, and also with a simpler analysis. Our algorithm works also in any metric space; however in this general setting, the running time remains quadratic.

Algorithm **A2**: Compute a diameter of the point set, and output the longest of the two stars centered at one of its endpoints.

Obviously the algorithm runs in $O(n \log n)$ time, with bottleneck being the diameter computation [15]. Let ab be a diameter pair, and assume w.l.o.g. that $|ab| = 1$. The ratio $1/2$ (or even $\frac{n}{2n-2}$) follows from the next lemma in conjunction with the obvious upper bound $L(T_{\text{OPT}}) \leq n$ (or $L(T_{\text{OPT}}) \leq n-1$).

Lemma 3.1. *Let S_a and S_b be the stars centered at the points a and b, respectively. Then $L(S_a) + L(S_b) \geq n$.*

Proof. Assume that $a = p_1$, $b = p_2$. For each $i = 3, \ldots, n$, the triangle inequality for the triple a, b, p_i gives

$$|ap_i| + |bp_i| \geq |ab| = 1.$$

By summing up we have

$$L(S_a) + L(S_b) = \sum_{i=3}^{n} (|ap_i| + |bp_i|) + 2|ab| \geq (n-2) + 2 = n.$$

■

We now continue with the new algorithm that achieves a (provable) $\frac{1}{2} + \frac{1}{500}$ approximation ratio within the same running time $O(n \log n)$.

Algorithm **A3**: Compute a diameter ab of the point set, and output the longest of the 5 non-crossing structures S_a, S_b, S_h, E_a, E_b, described below.

Assume w.l.o.g. that the ab is a horizontal unit segment, where $a = (0,0)$ and $b = (1,0)$. Let $h = (x_h, y_h)$ be a point in S with a largest value of $|y|$. By symmetry, we can assume that $y_h \geq 0$. S_a, S_b, and S_h are the 3 stars centered at a, b, and h respectively. E_a, resp. E_b, are two non-crossing extended stars centered at a, resp, b; details to follow. Each of the five structures can be computed in $O(n \log n)$ time, so the total execution time is also $O(n \log n)$.

Set $\delta = 0.05$, $w = 0.6$, $t = 0.6$ and $z = 0.48$, and refer to Fig. 2. Let ℓ_1, ℓ_2, ℓ_3, and ℓ_4, be four parallel vertical lines: $\ell_1 : x = 0$, $\ell_2 : x = 0.2$, $\ell_3 : x = 0.8$, $\ell_4 : x = 1$. Obviously, all points in S lie in the strip bounded by ℓ_1 and ℓ_4. Let V_m be the vertical parallel strip symmetric about the midpoint of ab and of width w. We refer to V_m as the middle strip; V_m is bounded by the vertical lines ℓ_2 and ℓ_3. Let V_a and V_b be the two vertical strips of width 0.2 bounded by ℓ_1 and ℓ_2, and by ℓ_3 and ℓ_4 respectively. Let $c = (x_c, y_c)$ be the intersection point between ℓ_3 and the circular arc γ_a of unit radius centered at a and sub-tending an angle of 60°. We have $x_c = 0.8$ and

$$y_c = \sqrt{1 - 0.8^2} = 0.6 = t.$$

We now describe the two extended star structures E_a and E_b. See also Fig. 3 for an example. To construct E_a, first compute the order of visibility of the points in V_b from point a by sorting. Then connect a with each point in the right strip V_b. Note that $b \in V_b$, thus $V_b \neq \emptyset$. Call S'_a the resulting star. The edges of this star together with the vertical line ℓ_3 divide $V_a \cup V_m$ into convex regions (wedges with a common apex a) ordered top-down. The subset of points in each wedge can be computed using binary search in overall $O(n \log n)$

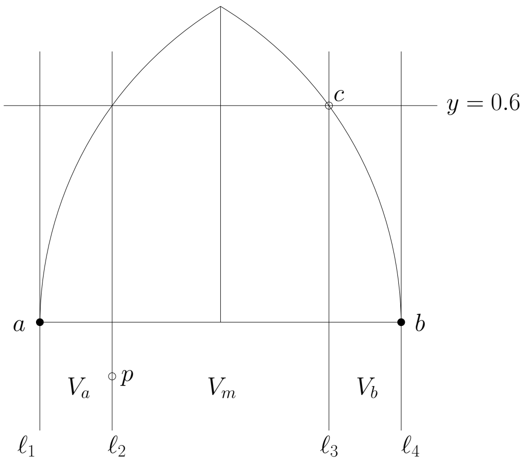


Figure 2: A diameter pair a, b at unit distance, and the three vertical strips V_a, V_m, and V_b. The two circular arcs γ_a and γ_b of unit radius centered at a and b intersect at the point $(1/2, \sqrt{3}/2)$. All points of S above ab lie in the region bounded by ab, γ_a and γ_b.

time (over all wedges). S'_a is extended (augmented) as follows. In each wedge, say paq, all points are connected either to a or to p, depending on the best (longest) overall connection cost. We denote the resulting *extended star* structure by E_a. The construction of E_b is analogous. It is clear by construction that both E_a and E_b are non-crossing.

Lemma 3.2. *For each $p \in S$, let $d_{max}(p)$ denote the maximum distance from p to other points in S. Then*

$$L(T_{OPT}) \leq \left[\sum_{i=1}^{n} d_{max}(p_i)\right] - 1.$$

Proof. Consider T_{OPT} rooted at a and drawn as an abstract tree with the root at the top in the usual manner. Let $\pi(v)$ denote the parent of a (non-root) vertex v. Uniquely assign each edge $\pi(v)v$ of T_{OPT} to vertex v. Obviously, $L(\pi(v)v) \leq d_{\max}(v)$ holds for each edge in the tree. By adding up the above inequalities, and taking into account that $d_{\max}(a) = |ab| = 1$, the lemma follows. ∎

Lemma 3.3. *Assume that $\sum_{i=1}^{n} |y_i| \geq \delta n$ for some positive constant $\delta \leq 1$. Then*

$$L(S_a) + L(S_b) \geq 2n\sqrt{\frac{1}{4} + \delta^2}.$$

Lemma 3.4. *Let n_a and n_b denote the number of points in the left and right vertical strips V_a and V_b. Then $L(E_a) \geq \frac{1+w}{4}(n + n_b)$, and similarly $L(E_b) \geq \frac{1+w}{4}(n + n_a)$. Consequently $L(E_a) + L(E_b) \geq \frac{1+w}{4}(2n + n_a + n_b)$. E_a and E_b can be constructed in $O(n \log n)$ time.*

Proof. The distance between ℓ_1 and ℓ_3 is $\frac{1+w}{2}$. By an argument similar to that in the proof of Lemma 3.1, the connection cost for a wedge with m points is at least $\frac{1+w}{4}m$. Therefore the total length of E_a is

$$L(E_a) \geq \frac{1+w}{2}n_b + \frac{1+w}{4}(n - n_b) = \frac{1+w}{4}(n + n_b).$$

The estimation of $L(E_b)$ is analogous. The running time has been established previously. ∎

Lemma 3.5. *Assume that* $\sum_{i=1}^{n} |y_i| \leq \delta n$ *and* $y_h \geq t$. *Then* $L(S_h) \geq (t - \delta)n$.

Proof.

$$L(S_h) \geq \sum_{i=1}^{n}(y_h - y_i) = ny_h - \sum_{i=1}^{n} y_i \geq ny_h - \sum_{i=1}^{n} |y_i| \geq ny_h - \delta n \geq (t - \delta)n. \qquad \blacksquare$$

Lemma 3.6. *Assume that* $|y_h| \leq t = 0.6$. *Let* $p \in S$ *be a point in the middle strip* V_m, *with* y*-coordinate satisfying* $|y| \leq 0.15$. *Then* $d_{max}(p) \leq 0.9605$.

Proof. It is straightforward to check that the maximum distance is attained for a point p on ℓ_2 with y-coordinate -0.15. The furthest point from p in the allowed region is c. Hence

$$d_{\max}(p) \leq |pc| = \sqrt{w^2 + (0.15 + t)^2} = \sqrt{0.6^2 + 0.75^2} \leq 0.9605. \qquad \blacksquare$$

We now distinguish the following four cases to complete our estimation of the approximation ratio.

Case 1: $\sum_{i=1}^{n} |y_i| \geq \delta n$. The algorithm outputs[2] S_a or S_b. By Lemma 3.3, the approximation ratio is at least

$$\frac{L(S_a) + L(S_b)}{2L(T_{\text{OPT}})} \geq \sqrt{\frac{1}{4} + \delta^2} \geq 0.502.$$

Case 2: $\sum_{i=1}^{n} |y_i| \leq \delta n$ and $y_h \geq t$. The algorithm outputs S_h. By Lemma 3.5, the approximation ratio is at least $t - \delta = 0.55$.

Case 3: $\sum_{i=1}^{n} |y_i| \leq \delta n$ and $y_h \leq t$ and $n_a + n_b \geq (1 - z)n$. The algorithm outputs E_a or E_b. We only need the last inequality in estimating the length. By Lemma 3.4, the approximation ratio is at least

$$\frac{L(E_a) + L(E_b)}{2L(T_{\text{OPT}})} \geq \frac{1 + w}{4} \cdot \frac{2n + n_a + n_b}{2n} \geq \frac{(1 + w)(3 - z)}{8} = \frac{1.6 \cdot 2.52}{8} = 0.504.$$

Case 4: $\sum_{i=1}^{n} |y_i| \leq \delta n$ and $y_h \leq t$ and $n_a + n_b \leq (1 - z)n$. The algorithm outputs S_a or S_b. There are at least $zn = 0.48n$ points in the middle strip V_m. Observe that at most $n/3$ points in V_m have $|y_i| \geq 0.15$; otherwise we would have

$$\sum_{i=1}^{n} |y_i| \geq \sum_{V_m} |y_i| > 0.15 \cdot \frac{n}{3} = 0.05n = \delta n,$$

a contradiction. It follows that at least $12n/25 - n/3 = 11n/75$ points in the middle strip have $|y_i| \leq 0.15$. By Lemma 3.2 and Lemma 3.6,

$$L(T_{\text{OPT}}) \leq \frac{64n}{75} + 0.9605 \cdot \frac{11n}{75} \leq 0.9943n.$$

The approximation ratio is at least

$$\frac{L(S_a) + L(S_b)}{2L(T_{\text{OPT}})} \geq \frac{n}{2 \cdot 0.9943n} \geq 0.502.$$

This completes the list of cases and thereby the proof of Theorem 1.2.

Remark. The example in Fig. 3 with n points (n even) equally spaced along a circle shows that the constant 0.502 measuring the approximation ratio achieved by our algorithm **A3**

[2] Here and in other instances it is meant that the algorithm outputs a structure at least as long as these.

cannot be improved to anything larger than $2/\pi$. Indeed the lengths of the five structures computed by the algorithm are $L(S_a) = L(S_b) = L(S_h) = L(E_a) = L(E_b) = (1 - o(1))\frac{2}{\pi}n$, while $L(T_{\mathrm{OPT}}) \geq L(H_{\mathrm{OPT}}) = (1 - o(1))n$.

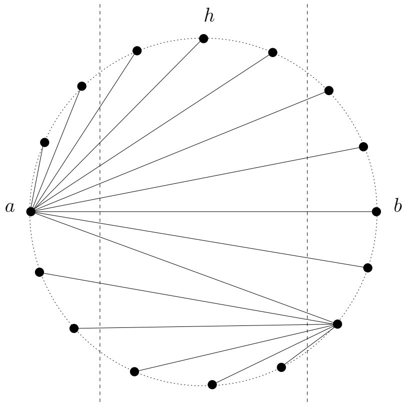


Figure 3: The non-crossing structure E_a for an example with $n = 16$ points on the circle. The middle strip V_m is bounded by the two dashed vertical lines.

4. The Hamiltonian cycle

In this section we present the proof of Theorem 1.3, which is similar (including notation) to that of Theorem 1.1. The rotated coordinate system Γ_α, and the x-coordinates x_i with respect to this system are denoted in the same way. By relabeling the points assume that the optimal cycle is $Q_{\mathrm{OPT}} = p_1, p_2, \ldots, p_n$ (with the convention that $p_{n+1} = p_1$). We approximate Q_{OPT} by constructing a non-crossing alternating path A on a subset of S, and then completing it to a non-crossing cycle using convex hull vertices. We need to observe that the alternating path A on the subset I of interior (non-hull) vertices of S produced by the algorithm of Abellanas et al. [1] is *not* good enough for *this* strategy: even though one endpoint of A (the first computed by the algorithm) is always on the convex hull of I, the other endpoint might be blocked by edges of A, so that A might not be extendible to a non-crossing Hamiltonian cycle (an example is shown in Fig. 4). Here, we give a stronger result that fits our purpose (for an even number of points).

Lemma 4.1. *Let $S = R \cup B$ with with $|R| = |B|$, be a linearly separable bipartition given by line ℓ. Then S admits an alternating non-crossing spanning path A such that (1) the edges of A cross ℓ at points ordered monotonically along ℓ; and (2) the two endpoints of A are incident to the two distinct edges of the convex hull that connect R and B (the two red-blue bridges). Such a Hamiltonian path can be computed in $O(n \log n)$ time. We refer to the underlying procedure as the* two-endpoint path construction algorithm.

Proof. We modify the algorithm of Abellanas et al. for path construction, so that the path is grown from the two endpoints and the two sub-paths merge "in the middle". Recall that $S = R \cup B$, and $|R| = |B|$, thus $|S|$ is even. Let $r_1 b_1$ and $r_2 b_2$ be the top and bottom red-blue edges of the convex hull conv(S), respectively, called *top* and *bottom* bridges; it is possible that $r_1 = r_2$ or $b_1 = b_2$ but not both. One endpoint of A is an endpoint of the top bridge, and the other endpoint of A is an endpoint of the bottom bridge, and they

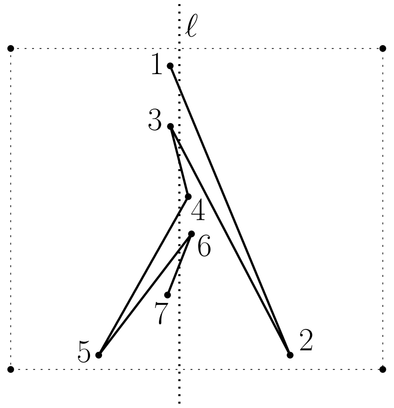


Figure 4: A non-crossing alternating path obtained by the algorithm of Abellanas et al. For the purpose of cycle construction, the path is non-extendible from its 2nd endpoint, vertex 7.

are chosen of opposite colors. Let $A = \{r_1, b_2\}$ or $A = \{b_1, r_2\}$ arbitrarily, containing two endpoints of the path. At every step, recompute the top and bottom bridges of $S \setminus A$, and append either the red or the blue vertex of each bridge to A such that the appended edges cross the separating line ℓ. In the last step, the convex hull of $S \setminus A$ is a red-blue segment that merges the two sub-paths. The two new edges added simultaneously at each step cannot cross each other; and they cannot cross previous edges, since they are separated from them by the convex hull of $S \setminus A$. Finally, they cannot extend the two sub-paths by the same point either, because $|S|$ is even. ∎

The next lemma follows from [10, Lemma 2.1]; we will only need its corollary, Lemma 4.3.

Lemma 4.2. ([10]). *Let $P = p_1, p_2, \ldots, p_n$ be a simple polygon (with the convention that $p_{n+1} = p_1$) and q be a point in the exterior of the convex hull of P, where $P \cup \{q\}$ is in general position. Then q sees one edge $p_i p_{i+1}$ of P. Such an edge can be found in $O(n)$ time.*

Lemma 4.3. *Let $P = p_1, p_2, \ldots, p_n$ be a simple polygon (with the convention that $p_{n+1} = p_1$) and q be a point in the exterior of the convex hull of P, where $P \cup \{q\}$ is in general position. Then the polygonal cycle P can be extended to include q so that $P \cup \{q\}$ is still a simple polygon. More precisely, there exists $i \in [n]$, so that $Q = p_1, \ldots, p_i, q, p_{i+1}, \ldots, p_n$ is a simple polygon. Moreover, $L(Q) > L(P)$. The extension can be computed in $O(n)$ time.*

Proof. By Lemma 4.2, q sees one edge $p_i p_{i+1}$ of P. Replacing this edge of P by the two edges $p_i q$ and $q p_{i+1}$ results in a simple polygon $Q = p_1, \ldots, p_i, q, p_{i+1}, \ldots, p_n$. By the triangle inequality, $L(Q) > L(P)$. The extension can be computed in $O(n)$ time, as determined by the time needed to find a visible edge. ∎

Note that the condition in the lemma that q lies in the exterior of the convex hull of P, is indeed necessary. Otherwise one cannot guarantee that q sees an edge of P.

(i) Let $S = S' \cup S''$, where S' is the set of convex hull vertices and S'' is the set of interior points. Let $S' = \{p_{j_1}, p_{j_2}, \ldots, p_{j_h}\}$. Put $h = |S'|$, $m = |S''|$, thus $n = h + m$. Assume first for simplicity that m is even. An easy modification of the algorithm, explained below, is used for m odd.

Algorithm **A4**:
Step 1. For all non-equivalent bisections of S'' (i.e., for all balanced bipartitions of S''): 1. Compute a non-crossing alternating path A by using the two-endpoint path construction algorithm (Lemma 4.1). 2. Extend A to a cycle by connecting its endpoints to (one or two) convex hull vertices. 3. Further extend this cycle to include the remaining hull vertices, by repeated invocation of Lemma 4.3.
Step 2. Output the longest such cycle (containing all points of S).

Observe that after Step 1.1, the two endpoints of the path are vertices of $\mathrm{conv}(S'')$, hence they can be connected to hull vertices to make a cycle. If m is odd, then there is a point $q \in S''$ on the line ℓ. Use the two-endpoint path construction algorithm for $S'' \setminus \{q\}$, and the same bisecting line ℓ. If q is in the interior of $\mathrm{conv}(S'' \setminus \{q\})$, then extend the path with point q, using Lemma 2.2. Otherwise, q sees the top or bottom bridge of $\mathrm{conv}(S'' \setminus \{q\})$, so the path can be extended by connecting q to the endpoint visible to q. The two endpoints of the extended path are on $\mathrm{conv}(S'')$, hence they can be connected to hull vertices to make a cycle, as in the case of even m.

References

[1] M. Abellanas, J. Garcia, G. Hernández, M. Noy, and P. Ramos: Bipartite embeddings of trees in the plane, *Discrete Applied Mathematics*, **93** (1999), 141–148.

[2] O. Aichholzer, S. Cabello, R. Fabila-Monroy, D. Flores-Peñaloza, T. Hackl, C. Huemer, F. Hurtado, and D. R. Wood: Edge-removal and non-crossing configurations in geometric graphs, *Proc. 24th European Workshop on Computational Geometry*, Nancy 2008, pp. 119–122.

[3] N. Alon, S. Rajagopalan and S. Suri: Long non-crossing configurations in the plane, *Fundamenta Informaticae* **22** (1995), 385–394. Also in *Proc. 9th ACM Sympos. on Comput. Geom.*, 1993, 257–263.

[4] M. Bern and D. Eppstein: Approximation algorithms for geometric problems, in *Approximation Algorithms for NP-hard Problems* (D. S. Hochbaum, editor), PWS, Boston, 1997, pp. 296–345.

[5] J. Černý, Z. Dvořák, V. Jelínek, and J. Kára: Noncrossing Hamiltonian paths in geometric graphs, *Discrete Applied Mathematics*, **155** (2007), 1096–1105.

[6] T. K. Dey: Improved bounds on planar k-sets and related problems, *Discrete & Computational Geometry*, **19** (1998), 373–382.

[7] H. Edelsbrunner: *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[8] D. Eppstein: Spanning trees and spanners, in *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, editors), Elsevier Science, Amsterdam, 2000, pp. 425–461.

[9] S. P. Fekete: Simplicity and hardness of the maximum traveling salesman problem under geometric distances, *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, 1999, pp. 337–345.

[10] F. Hurtado, M. Kano, D. Rappaport, and Cs. D. Tóth: Encompassing colored planar straight line graphs, *Computational Geometry: Theory and Applications*, **39** (1) (2008), 14–23.

[11] G. Károlyi, J. Pach and G. Tóth: Ramsey-type results for geometric graphs. I, *Discrete and Computational Geometry* **18** (1997), 247-255.

[12] G. Károlyi, J. Pach, G. Tóth and P. Valtr: Ramsey-type results for geometric graphs. II, *Discrete and Computational Geometry*, **20** (1998), 375–388.

[13] L. Lovász: On the number of halving lines, *Ann. Univ. Sci. Budapest, Eötvös, Sec. Math.*, **14** (1971), 107–108.

[14] J. S. B. Mitchell: Geometric shortest paths and network optimization, in *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, editors), Elsevier Science, Amsterdam, 2000, pp. 633–701.

[15] F. Preparata and M. Shamos: *Computational Geometry: An Introduction*, Springer, New York, 1985.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 323-334
www.stacs-conf.org

THE COMPLEXITY OF APPROXIMATING BOUNDED-DEGREE BOOLEAN #CSP

MARTIN DYER[1] AND LESLIE ANN GOLDBERG[2] AND MARKUS JALSENIUS[2,3] AND DAVID RICHERBY[1]

[1] School of Computing, University of Leeds, Leeds, LS2 9JT, U.K.
E-mail address: {M.E.Dyer,D.M.Richerby}@leeds.ac.uk

[2] Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, U.K.
E-mail address: L.A.Goldberg@liverpool.ac.uk

[3] Current address: Department of Computer Science, University of Bristol, Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB, U.K.
E-mail address: M.Jalsenius@bristol.ac.uk

ABSTRACT. The degree of a CSP instance is the maximum number of times that a variable may appear in the scope of constraints. We consider the approximate counting problem for Boolean CSPs with bounded-degree instances, for constraint languages containing the two unary constant relations $\{0\}$ and $\{1\}$. When the maximum degree is at least 25 we obtain a complete classification of the complexity of this problem. It is exactly solvable in polynomial-time if every relation in the constraint language is affine. It is equivalent to the problem of approximately counting independent sets in bipartite graphs if every relation can be expressed as conjunctions of $\{0\}$, $\{1\}$ and binary implication. Otherwise, there is no FPRAS unless $\mathbf{NP} = \mathbf{RP}$. For lower degree bounds, additional cases arise in which the complexity is related to the complexity of approximately counting independent sets in hypergraphs.

1. Introduction

In the constraint satisfaction problem (CSP), we seek to assign values from some domain to a set of variables, while satisfying given constraints on the combinations of values that certain subsets of the variables may take. Constraint satisfaction problems are ubiquitous in computer science, with close connections to graph theory, database query evaluation, type inference, satisfiability, scheduling and artificial intelligence [20, 22, 25]. CSP can also be reformulated in terms of homomorphisms between relational structures [14] and conjunctive query containment in database theory [20]. Weighted versions of CSP appear in statistical physics, where they correspond to partition functions of spin systems [31].

1998 ACM Subject Classification: F.2.2, G.2.1.

Key words and phrases: Boolean constraint satisfaction problem, generalized satisfiability, counting, approximation algorithms.

Funded in part by the EPSRC grant "The Complexity of Counting in Constraint Satisfaction Problems".

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2466

We give formal definitions in Section 2 but, for now, consider an undirected graph G and the CSP where the domain is $\{\text{red}, \text{green}, \text{blue}\}$, the variables are the vertices of G and the constraints specify that, for every edge $xy \in G$, x and y must be assigned different values. Thus, in a satisfying assignment, no two adjacent vertices are given the same colour: the CSP is satisfiable if, and only if, the graph is 3-colourable. As a second example, given a formula in 3-CNF, we can write a system of constraints over the variables, with domain $\{\text{true}, \text{false}\}$, that requires the assignment to each clause to satisfy at least one literal. Clearly, the resulting CSP is directly equivalent to the original satisfiability problem.

1.1. Decision CSP

In the *uniform constraint satisfaction problem*, we are given the set of constraints explicitly, as lists of allowable combinations for given subsets of the variables; these lists can be considered as relations over the domain. Since it includes problems such as 3-SAT and 3-COLOURABILITY, uniform CSP is **NP**-complete. However, uniform CSP also includes problems in **P**, such as 2-SAT and 2-COLOURABILITY, raising the natural question of what restrictions lead to tractable problems. There are two natural ways to restrict CSP: we can restrict the form of the instances and we can restrict the form of the constraints.

The most common restriction to CSP is to allow only certain fixed relations in the constraints. The list of allowed relations is known as the *constraint language* and we write CSP(Γ) for the so-called *non-uniform* CSP in which each constraint states that the values assigned to some tuple of variables must be a tuple in a specified relation in Γ.

The classic example of this is Schaefer's dichotomy for Boolean constraint languages Γ (i.e., those with domain $\{0, 1\}$; often called "generalized satisfiability") [26]. He showed that CSP(Γ) is in **P** if Γ is included in one of six classes and is **NP**-complete, otherwise. More recently, Bulatov has produced a corresponding dichotomy for the three-element domain [2]. These two results restrict the size of the domain but allow relations of arbitrary arity in the constraint language. The converse restriction — relations of restricted arity, especially binary relations, over arbitrary finite domains — has also been studied in depth [16, 17].

For all Γ studied so far, CSP(Γ) has been either in **P** or **NP**-complete and Feder and Vardi have conjectured that this holds for every constraint language [14]. Ladner has shown that it is not the case that every problem in **NP** is either in **P** or **NP**-complete since, if $\mathbf{P} \neq \mathbf{NP}$, there is an infinite, strict hierarchy between the two [23]. However, there are problems in **NP**, such as graph Hamiltonicity and even connectedness, that cannot be expressed as CSP(Γ) for any finite Γ[1] and Ladner's diagonalization does not seem to be expressible in CSP [14], so a dichotomy for CSP appears possible.

Restricting the tree-width of instances has also been a fruitful direction of research [15, 21]. In contrast, little is known about restrictions on the degree of instances, i.e., the maximum number of times that any variable may appear. Dalmau and Ford have shown that, for any fixed Boolean constraint language Γ containing the constant unary relations $R_{\text{zero}} = \{0\}$ and $R_{\text{one}} = \{1\}$, the complexity of CSP(Γ) for instances of degree at most three is exactly the same as the complexity of CSP(Γ) with no degree restriction [6]. The case where variables may appear at most twice has not yet been completely classified; it is known that degree-2 CSP(Γ) is as hard as general CSP(Γ) whenever Γ contains R_{zero} and R_{one} and some relation that is not a Δ-matroid [13]; the known polynomial-time cases come from restrictions on the kinds of Δ-matroids that appear in Γ [6].

[1] This follows from results on the expressive power of existential monadic second-order logic [12].

1.2. Counting CSP

A generalization of classical CSP is to ask how many satisfying solutions there are. This is referred to as counting CSP, #CSP. Clearly, the decision problem is reducible to counting: if we can efficiently count the solutions, we can efficiently determine whether there is at least one. The converse does not hold: for example, we can determine in polynomial time whether a graph admits a perfect matching but it is #**P**-complete to count the perfect matchings, even in a bipartite graph [29].

#**P** is the class of functions f for which there is a nondeterministic, polynomial-time Turing machine that has exactly $f(x)$ accepting paths for input x [28]. It is easily seen that the counting version of any **NP** decision problem is in #**P** and #**P** can be considered the counting "analogue" of **NP**. Note, though that problems that are #**P**-complete under appropriate reductions are, under standard complexity-theoretic assumptions, considerably harder than **NP**-complete problems: $\mathbf{P}^{\#\mathbf{P}}$ includes the whole of the polynomial hierarchy [27], whereas $\mathbf{P}^{\mathbf{NP}}$ is generally thought not to.

Although no dichotomy is known for CSP, Bulatov has recently shown that, for all Γ, #CSP(Γ) is either computable in polynomial time or #**P**-complete [3]. However, Bulatov's dichotomy sheds little light on which constraint languages yield polynomial-time counting CSPs and which do not. The criterion of the dichotomy is based on "defects" in a certain infinite algebra built up from the polymorphisms of Γ and it is open whether the characterization is even decidable. It also seems not to apply to bounded-degree #CSP.

So, although there is a full dichotomy for #CSP(Γ), results for restricted forms of constraint language are still of interest. Creignou and Hermann have shown that only one of Schaefer's polynomial-time cases for Boolean languages survives the transition to counting: #CSP(Γ) $\in$ **FP** (i.e., has a polynomial time algorithm) if Γ is affine (i.e., each relation is the solution set of a system of linear equations over GF_2) and is #**P**-complete, otherwise [5]. This result has been extended to rational and even complex-weighted instances [4,10] and, in the latter case, the dichotomy is shown to hold for the restriction of the problem in which instances have degree 3. This implies that the degree-3 problem #$\mathrm{CSP}_3(\Gamma)$ (#CSP(Γ) restricted to instances of degree 3) is in **FP** if Γ is affine and is #**P**-complete, otherwise.

1.3. Approximate counting

Since #CSP(Γ) is very often #**P**-complete, approximation algorithms play an important role. The key concept is that of a *fully polynomial randomized approximation scheme* (FPRAS). This is a randomized algorithm for computing some function $f(x)$, taking as its input x and a constant $\epsilon > 0$, and computing a value Y such that $e^{-\epsilon} \leqslant Y/f(x) \leqslant e^{\epsilon}$ with probability at least $\frac{3}{4}$, in time polynomial in both $|x|$ and ϵ^{-1}. (See Section 2.4.)

Dyer, Goldberg and Jerrum have classified the complexity of approximately computing #CSP(Γ) for Boolean constraint languages [9]. When all relations in Γ are affine, #CSP(Γ) can be computed exactly in polynomial time by the result of Creignou and Hermann discussed above [5]. Otherwise, if every relation in Γ can be defined by a conjunction of pins (i.e., assertions $v = 0$ or $v = 1$) and Boolean implications, then #CSP(Γ) is as hard to approximate as the problem #BIS of counting independent sets in a bipartite graph; otherwise, #CSP(Γ) is as hard to approximate as the problem #SAT of counting the satisfying truth assignments of a Boolean formula. Dyer, Goldberg, Greenhill and Jerrum have shown that the latter problem is complete for #**P** under appropriate approximation-preserving reductions (see Section 2.4) and has no FPRAS unless **NP** = **RP** [8], which is thought to be unlikely. The complexity of #BIS is currently open: there is no known FPRAS but it is not known to be #**P**-complete, either. #BIS is known to be complete for a logically-defined subclass of #**P** with respect to approximation-preserving reductions [8].

1.4. Our result

We consider the complexity of approximately solving Boolean #CSP problems when instances have bounded degree. Following Dalmau and Ford [6] and Feder [13] we consider the case in which $R_{\text{zero}} = \{0\}$ and $R_{\text{one}} = \{1\}$ are available. We proceed by showing that any Boolean relation that is not definable as a conjunction of ORs or NANDs can be used in low-degree instances to assert equalities between variables. Thus, we can side-step degree restrictions by replacing high-degree variables with distinct variables asserted to be equal.

Our main result, Corollary 6.6, is a trichotomy for the case in which instances have maximum degree d for some $d \geqslant 25$. If every relation in Γ is affine, then $\#\text{CSP}_d(\Gamma \cup \{R_{\text{zero}}, R_{\text{one}}\})$ is solvable in polynomial time. Otherwise, if every relation in Γ can be defined as a conjunction of R_{zero}, R_{one} and binary implications, then $\#\text{CSP}_d(\Gamma \cup \{R_{\text{zero}}, R_{\text{one}}\})$ is equivalent in approximation complexity to #BIS. Otherwise, it has no FPRAS unless **NP** = **RP**. Theorem 6.5 gives a partial classification of the complexity when $d < 25$. In the new cases that arise here, the complexity is given in terms of the complexity of counting independent sets in hypergraphs with bounded degree and bounded hyper-edge size. The complexity of this problem is not fully understood and we explain what is known about it in Section 6.

2. Preliminaries

2.1. Basic notation

We write $\overline{a}$ for the tuple $\langle a_1, \ldots, a_r \rangle$, which we often shorten to $\overline{a} = a_1 \ldots a_r$. We write a^r for the r-tuple $a \ldots a$ and $\overline{a}\overline{b}$ for the tuple formed from the elements of $\overline{a}$ followed by those of $\overline{b}$. The *bit-wise complement* of a relation $R \subseteq \{0,1\}^r$ is the relation $\widetilde{R} = \{\langle a_1 \oplus 1, \ldots, a_r \oplus 1 \rangle \mid \overline{a} \in R\}$, where $\oplus$ denotes addition modulo 2.

We say that a relation R is *ppp-definable*[2] in a relation R' and write $R \leqslant_{\text{ppp}} R'$ if R can be obtained from R' by some sequence of the following operations:

- permutation of columns (for notational convenience only);
- pinning (taking sub-relations of the form $R_{i \mapsto c} = \{\overline{a} \in R \mid a_i = c\}$ for some i and some $c \in \{0,1\}$); and
- projection ("deleting the ith column" to give the relation $\{a_1 \ldots a_{i-1} a_{i+1} \ldots a_r \mid a_1 \ldots a_r \in R\}$).

It is easy to see that $\leqslant_{\text{ppp}}$ is reflexive and transitive and that, if $R \leqslant_{\text{ppp}} R'$, then R can be obtained from R' by first permuting the columns, then making some pins and then projecting.

We write $R_{=} = \{00, 11\}$, $R_{\neq} = \{01, 10\}$, $R_{\text{OR}} = \{01, 10, 11\}$, $R_{\text{NAND}} = \{00, 01, 10\}$, $R_{\rightarrow} = \{00, 01, 11\}$ and $R_{\leftarrow} = \{00, 10, 11\}$. For $k \geqslant 2$, we write $R_{=,k} = \{0^k, 1^k\}$, $R_{\text{OR},k} = \{0,1\}^k \setminus \{0^k\}$ and $R_{\text{NAND},k} = \{0,1\}^k \setminus \{1^k\}$ (i.e., k-ary equality, OR and NAND).

[2] This should not be confused with the concept of primitive positive definability (pp-definability) which appears in algebraic treatments of CSP and #CSP, for example in the work of Bulatov [3].

2.2. Boolean constraint satisfaction problems

A *constraint language* is a set $\Gamma = \{R_1, \dots, R_m\}$ of named Boolean relations. Given a set V of variables, the set of *constraints* over Γ is the set $\mathrm{Cons}(V, \Gamma)$ which contains $R(\overline{v})$ for every relation $R \in \Gamma$ with arity r and every $\overline{v} \in V^r$. Note that $v = v'$ and $v \neq v'$ are not constraints unless the appropriate relations are included in Γ. The *scope* of a constraint $R(\overline{v})$ is the tuple $\overline{v}$, which need not consist of distinct variables.

An *instance* of the constraint satisfaction problem (CSP) over Γ is a set V of variables and a set $C \subseteq \mathrm{Cons}(V, \Gamma)$ of constraints. An *assignment* to a set V of variables is a function $\sigma\colon V \to \{0, 1\}$. An assignment to V *satisfies* an instance (V, C) if $\langle\, \sigma(v_1), \dots, \sigma(v_r)\,\rangle \in R$ for every constraint $R(v_1, \dots, v_r)$. We write $Z(I)$ for the number of satisfying assignments to a CSP instance I. We study the counting CSP problem $\#\mathrm{CSP}(\Gamma)$, parameterized by Γ, in which we must compute $Z(I)$ for an instance $I = (V, C)$ of CSP over Γ.

The *degree* of an instance is the greatest number of times any variable appears among its constraints. Note that the variable v appears twice in the constraint $R(v, v)$. Our specific interest in this paper is in classifying the complexity of bounded-degree counting CSPs. For a constraint language Γ and a positive integer d, define $\#\mathrm{CSP}_d(\Gamma)$ to be the restriction of $\#\mathrm{CSP}(\Gamma)$ to instances of degree at most d. Instances of degree 1 are trivial.

Theorem 2.1. *For any* Γ, $\#\mathrm{CSP}_1(\Gamma) \in \mathbf{FP}$. ■

When considering $\#\mathrm{CSP}_d$ for $d \geqslant 2$, we follow established practice by allowing *pinning* in the constraint language [6, 13]. We write $R_{\mathrm{zero}} = \{0\}$ and $R_{\mathrm{one}} = \{1\}$ for the two singleton unary relations. We refer to constraints in R_{zero} and R_{one} as *pins*. To make notation easier, we will sometimes write constraints using constants instead of explicit pins. That is, we will allow the constants 0 and 1 to appear in the place of variables in the scopes of constraints. Such constraints can obviously be rewritten as a set of "proper" constraints, without increasing degree. We let Γ_{pin} denote the constraint language $\{R_{\mathrm{zero}}, R_{\mathrm{one}}\}$.

2.3. Hypergraphs

A *hypergraph* $H = (V, E)$ is a set $V = V(H)$ of vertices and a set $E = E(H) \subseteq \mathcal{P}(V)$ of non-empty *hyper-edges*. The *degree* of a vertex $v \in V(H)$ is the number $d(v) = |\{e \in E(H) \mid v \in e\}|$ and the degree of a hypergraph is the maximum degree of its vertices. If $w = \max\{|e| \mid e \in E(H)\}$, we say that H has *width* w. An *independent set* in a hypergraph H is a set $S \subseteq V(H)$ such that $e \nsubseteq S$ for every $e \in E(H)$. Note that an independent set may contain more than one vertex from any hyper-edge of size at least three.

We write $\#w$-HIS for the problem of counting the independent sets in a width-w hypergraph H, and $\#w\text{-HIS}_d$ for the restriction of $\#w$-HIS to inputs of degree at most d.

2.4. Approximation complexity

A *randomized approximation scheme* (RAS) for a function $f\colon \Sigma^* \to \mathbb{N}$ is a probabilistic Turing machine that takes as input a pair $(x, \epsilon) \in \Sigma^* \times (0, 1)$, and produces, on an output tape, an integer random variable Y with $\Pr(e^{-\epsilon} \leqslant Y/f(x) \leqslant e^{\epsilon}) \geqslant \frac{3}{4}$.[3] A *fully polynomial randomized approximation scheme (FPRAS)* is a RAS that runs in time $\mathrm{poly}(|x|, \epsilon^{-1})$.

[3] The choice of the value $\frac{3}{4}$ is inconsequential: the same class of problems has an FPRAS if we choose any probability p with $\frac{1}{2} < p < 1$ [18].

To compare the complexity of approximate counting problems, we use the AP-reductions of [8]. Suppose f and g are two functions from some input domain Σ^* to the natural numbers and we wish to compare the complexity of approximately computing f to that of approximately computing g. An *approximation-preserving* reduction from f to g is a probabilistic oracle Turing machine M that takes as input a pair $(x, \epsilon) \in \Sigma^* \times (0, 1)$, 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 $\delta^{-1} \leqslant \mathrm{poly}(|x|, \epsilon^{-1})$; (ii) M is a randomized approximation scheme for f whenever the oracle is a randomized approximation scheme for g; and (iii) the run-time of M is polynomial in $|x|$ and ϵ^{-1}.

If there is an approximation-preserving reduction from f to g, we write $f \leqslant_{\mathrm{AP}} g$ and say that f is *AP-reducible* to g. If g has an FPRAS, then so does f. If $f \leqslant_{\mathrm{AP}} g$ and $g \leqslant_{\mathrm{AP}} f$, then we say that f and g are *AP-interreducible* and write $f \equiv_{\mathrm{AP}} g$.

3. Classes of relations

A relation $R \subseteq \{0, 1\}^r$ is *affine* if it is the set of solutions to some system of linear equations over GF_2. That is, there is a set Σ of equations in variables $x_1, \ldots, x_r$, each of the form $x_{i_1} \oplus \cdots \oplus x_{i_n} = c$, where $\oplus$ denotes addition modulo 2 and $c \in \{0, 1\}$, such that $\overline{a} \in R$ if, and only if, the assignment $x_1 \mapsto a_1, \ldots, x_r \mapsto a_r$ satisfies every equation in Σ. Note that the empty and complete relations are affine.

We define IM-conj to be the class of relations defined by a conjunction of pins and (binary) implications. This class is called IM_2 in [9].

Lemma 3.1. *If $R \in$ IM-conj is not affine, then $R_{\rightarrow} \leqslant_{\mathrm{ppp}} R$.* ■

Let OR-conj be the set of Boolean relations that are defined by a conjunction of pins and ORs of any arity and NAND-conj the set of Boolean relations definable by conjunctions of pins and NANDs (i.e., negated conjunctions) of any arity. We say that one of the defining formulae of these relations is *normalized* if no pinned variable appears in any OR or NAND, the arguments of each individual OR and NAND are distinct, every OR or NAND has at least two arguments and no OR or NAND's arguments are a subset of any other's.

Lemma 3.2. *Every* OR-*conj (respectively,* NAND-*conj) relation is defined by a unique normalized formula.* ■

Given the uniqueness of defining normalized formulae, we define the *width* of an OR-conj or NAND-conj relation R to be $\mathrm{wd}(R)$, the greatest number of arguments to any of the ORs or NANDs in the normalized formula that defines it. Note that, from the definition of normalized formulae, there are no relations of width 1.

Lemma 3.3. *If $R \in$* OR-*conj has width w, then $R_{\mathrm{OR},2}, \ldots, R_{\mathrm{OR},w} \leqslant_{\mathrm{ppp}} R$. Similarly, if $R \in$* NAND-*conj has width w, then $R_{\mathrm{NAND},2}, \ldots, R_{\mathrm{NAND},w} \leqslant_{\mathrm{ppp}} R$.* ■

Given tuples $\overline{a}, \overline{b} \in \{0, 1\}^r$, we write $\overline{a} \leqslant \overline{b}$ if $a_i \leqslant b_i$ for all $i \in [1, r]$. If $\overline{a} \leqslant \overline{b}$ and $\overline{a} \neq \overline{b}$, we write $\overline{a} < \overline{b}$. We say that a relation $R \subseteq \{0, 1\}^r$ is *monotone* if, whenever $\overline{a} \in R$ and $\overline{a} \leqslant \overline{b}$, then $\overline{b} \in R$. We say that R is *antitone* if, whenever $\overline{a} \in R$ and $\overline{b} \leqslant \overline{a}$, then $\overline{b} \in R$. Clearly, R is monotone if, and only if, $\widetilde{R}$ is antitone. Call a relation *pseudo-monotone* (respectively, *pseudo-antitone*) if its restriction to non-constant columns is monotone (respectively, antitone). The following is a consequence of results in [19, Chapter 7.1.1].

Proposition 3.4. *A relation $R \subseteq \{0, 1\}^r$ is in* OR-*conj (respectively,* NAND-*conj) if, and only if, it is pseudo-monotone (respectively, pseudo-antitone).* ■

4. Simulating equality

An important ingredient in bounded-degree dichotomy theorems [4] is expressing equality using constraints from a language that does not necessarily include the equality relation.

A constraint language Γ is said to *simulate* the k-ary equality relation $R_{=,k}$ if, for some $\ell \geqslant k$, there is a $(\Gamma \cup \Gamma_{\mathrm{pin}})$-CSP instance I with variables $x_1, \dots, x_\ell$ that has exactly $m \geqslant 1$ satisfying assignments σ with $\sigma(x_1) = \dots = \sigma(x_k) = 0$, exactly m with $\sigma(x_1) = \dots = \sigma(x_k) = 1$ and no other satisfying assignments. If, further, the degree of I is d and the degree of each variable $x_1, \dots, x_k$ is at most $d-1$, we say that Γ *d-simulates* $R_{=,k}$. We say that Γ *d-simulates equality* if it d-simulates $R_{=,k}$ for all $k \geqslant 2$.

The point is that, if Γ d-simulates equality, we can express the constraint $y_1 = \dots = y_r$ in $\Gamma \cup \Gamma_{\mathrm{pin}}$ and then use each y_i in one further constraint, while still having an instance of degree d. The variables $x_{k+1}, \dots, x_\ell$ in the definition function as auxiliary variables and are not used in any other constraint. Simulating equality makes degree bounds moot.

Proposition 4.1. *If Γ d-simulates equality, then $\#\mathrm{CSP}(\Gamma) \leqslant_{\mathrm{AP}} \#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}})$.* ∎

We now investigate which relations simulate equality.

Lemma 4.2. *$R \in \{0,1\}^r$ 3-simulates equality if $R_= \leqslant_{\mathrm{ppp}} R$, $R_{\neq} \leqslant_{\mathrm{ppp}} R$ or $R_{\rightarrow} \leqslant_{\mathrm{ppp}} R$.*

Proof. For each $k \geqslant 2$, we show how to 3-simulate $R_{=,k}$. We may assume without loss of generality that the ppp-definition of $R_=$, $R_{\neq}$ or $R_{\rightarrow}$ from R involves applying the identity permutation to the columns, pinning columns 3 to $3+p-1$ inclusive to zero, pinning columns $3+p$ to $3+p+q-1$ inclusive to one (that is, pinning $p \geqslant 0$ columns to zero and $q \geqslant 0$ to one) and then projecting away all but the first two columns.

Suppose first that $R_= \leqslant_{\mathrm{ppp}} R$ or $R_{\rightarrow} \leqslant_{\mathrm{ppp}} R$. R must contain $\alpha \geqslant 1$ tuples that begin 000^p1^q, $\beta \geqslant 0$ that begin 010^p1^q and $\gamma \geqslant 1$ that begin 110^p1^q, with $\beta = 0$ unless we are ppp-defining $R_{\rightarrow}$. We consider, first, the case where $\alpha = \gamma$, and show that we can 3-simulate $R_{=,k}$, expressing the constraint $R_{=,k}(x_1, \dots, x_k)$ with the constraints

$$R(x_1x_20^p1^q*),\ R(x_2x_30^p1^q*), \dots,\ R(x_{k-1}x_k0^p1^q*),\ R(x_kx_10^p1^q*)\,,$$

where $*$ denotes a fresh $(r-2-p-q)$-tuple of variables in each constraint. These constraints are equivalent to $x_1 = \dots = x_k = x_1$ or to $x_1 \rightarrow \dots \rightarrow x_k \rightarrow x_1$ so constrain the variables $x_1, \dots, x_k$ to have the same value, as required. Every variable appears at most twice and there are α^k solutions to these constraints that put $x_1 = \dots = x_k = 0$, $\gamma^k = \alpha^k$ solutions with $x_1 = \dots = x_k = 1$ and no other solutions. Hence, R 3-simulates $R_{=,k}$, as required.

We now show, by induction on r, that we can 3-simulate $R_{=,k}$ even in the case that $\alpha \neq \gamma$. For the base case, $r = 2$, we have $\alpha = \gamma = 1$ and we are done. For the inductive step, let $r > 2$ and assume, w.l.o.g. that $\alpha > \gamma$ ($\alpha < \gamma$ is symmetric). In particular, we have $\alpha \geqslant 2$, so there are distinct tuples $000^p1^q\overline{a}$, and $000^p1^q\overline{b}$ and $110^p1^q\overline{c}$ in R. Choose j such that $a_j \neq b_j$. Pinning the $(2+p+q+j)$th column of R to c_j and projecting out the resulting constant column gives a relation R' of arity $r-1$ containing at least one tuple beginning 000^p1^q and at least one beginning 110^p1^q: by the inductive hypothesis, R' 3-simulates $R_{=,k}$.

Finally, we consider the case that $R_{\neq} \leqslant_{\mathrm{ppp}} R$. R contains $\alpha \geqslant 1$ tuples beginning 010^p1^q and $\beta \geqslant 1$ beginning 100^p1^q. We express the constraint $R_{=,k}(x_1, \dots, x_k)$ by introducing fresh variables $y_1, \dots, y_k$ and using the constraints

$$\begin{array}{llcll} R(x_1y_10^p1^q*), & R(x_2y_20^p1^q*), & \dots, & R(x_{k-1}y_{k-1}0^p1^q*), & R(x_ky_k0^p1^q*),\\ R(y_1x_20^p1^q*), & R(y_2x_30^p1^q*), & \dots, & R(y_{k-1}x_k0^p1^q*), & R(y_kx_10^p1^q*)\,. \end{array}$$

There are $\alpha^k\beta^k$ solutions when $x_1 = \cdots = x_k = 0$ (and $y_1 = \cdots = y_k = 1$) and $\beta^k\alpha^k$ solutions when the xs are 1 and the ys are 0. There are no other solutions and no variable is used more than twice. ■

For $c \in \{0,1\}$, an r-ary relation is *c-valid* if it contains the tuple c^r.

Lemma 4.3. *Let $r \geqslant 2$ and let $R \subseteq \{0,1\}^r$ be 0- and 1-valid but not complete. Then R 3-simulates equality.* ■

In the following lemma, we do not require R and R' to be distinct. The technique is to assert $x_1 = \cdots = x_k$ by simulating the formula $\mathrm{OR}(x_1,y_1) \wedge \mathrm{NAND}(y_1,x_2) \wedge \mathrm{OR}(x_2,y_2) \wedge \mathrm{NAND}(y_2,x_3) \wedge \cdots \wedge \mathrm{OR}(x_k,y_k) \wedge \mathrm{NAND}(y_k,x_1)$.

Lemma 4.4. *If $R_{\mathrm{OR}} \leqslant_{\mathrm{ppp}} R$ and $R_{\mathrm{NAND}} \leqslant_{\mathrm{ppp}} R'$, then $\{R,R'\}$ 3-simulates equality.* ■

5. Classifying relations

We are now ready to prove that every Boolean relation R is in OR-conj, in NAND-conj or 3-simulates equality. If R_0 and R_1 are r-ary, let $R_0 + R_1 = \{0\overline{a} \mid \overline{a} \in R_0\} \cup \{1\overline{a} \mid \overline{a} \in R_1\}$.

Lemma 5.1. *Let $R_0, R_1 \in$ OR-conj and let $R = R_0 + R_1$. Then $R \in$ OR-conj, $R \in$ NAND-conj or R 3-simulates equality.*

Proof. Let R_0 and R_1 have arity r. We may assume that R has no constant columns. If it does, let R' be the relation that results from projecting them away. $R' = R'_0 + R'_1$, where both R'_0 and R'_1 are OR-conj relations. By the remainder of the proof, $R' \in$ OR-conj, $R' \in$ NAND-conj or R' 3-simulates equality. Re-instating the constant columns does not alter this. For R without constant columns, there are two cases.

Case 1. $R_0 \subseteq R_1$. Suppose R_i is defined by the normalized OR-conj formula ϕ_i in variables $x_2, \dots, x_{r+1}$. Then R is defined by the formula

$$\phi_0 \vee (x_1 = 1 \wedge \phi_1) \equiv (\phi_0 \vee x_1 = 1) \wedge (\phi_0 \vee \phi_1) \equiv (\phi_0 \vee x_1 = 1) \wedge \phi_1\,, \tag{5.1}$$

where the second equivalence is because ϕ_0 implies ϕ_1, because $R_0 \subseteq R_1$. R_1 has no constant column, since such a column would have to be constant with the same value in R_0, contradicting our assumption that R has no constant columns. There are two cases.

Case 1.1. *R_0 has no constant columns.* $x_1 = 1$ is equivalent to $\mathrm{OR}(x_1)$ and ϕ_0 contains no pins, so we can rewrite $\phi_0 \vee x_1 = 1$ in CNF. Therefore, (5.1) is OR-conj.

Case 1.2. *R_0 has a constant column.* Suppose first that the kth column of R_0 is constant-zero. R_1 has no constant columns, so the projection of R onto its first and $(k+1)$st columns gives the relation $R_{\leftarrow}$, and R 3-simulates equality by Lemma 4.2. Otherwise, all constant columns of R_0 contain ones. Then ϕ_0 is in CNF, since every pin $x_i = 1$ in ϕ_0 can be written $\mathrm{OR}(x_i)$. Thus, we can write $\phi_0 \vee x_1 = 1$ in CNF, so (5.1) defines an OR-conj relation.

Case 2. $R_0 \nsubseteq R_1$. We will show that R 3-simulates equality or is in NAND-conj. We consider two cases (recall that no relation has width 1).

Case 2.1. *At least one of R_0 and R_1 has positive width.* There are two sub-cases.

Case 2.1.1. *R_1 has a constant column.* Suppose the kth column of R_1 is constant. If the kth column of R_0 is also constant, then the projection of R to its first and $(k+1)$st columns is either equality or disequality (since the corresponding column of R is not constant) so R 3-simulates equality by Lemma 4.2. Otherwise, if the projection of R to the first and $(k+1)$st columns is $R_{\rightarrow}$, then R 3-simulates equality by Lemma 4.2. Otherwise, that projection must be R_{NAND}. By Lemma 3.3 and the assumption of Case 2.1, R_{OR} is ppp-definable in at least one of R_0 and R_1 so R 3-simulates equality by Lemma 4.4.

Case 2.1.2. *R_1 has no constant columns.* By Proposition 3.4, R_1 is monotone. Let $\overline{a} \in R_0 \setminus R_1$: by applying the same permutation to the columns of R_0 and R_1, we may assume that $\overline{a} = 0^\ell 1^{r-\ell}$. We must have $\ell \geqslant 1$ as every non-empty r-ary monotone relation contains the tuple 1^r. Let $\overline{b} \in R_1$ be a tuple such that $a_i = b_i$ for a maximal initial segment of $[1,r]$. By monotonicity of R_1, we may assume that $\overline{b} = 0^k 1^{r-k}$. Further, we must have $k < \ell$, since, otherwise, we would have $\overline{b} < \overline{a}$, contradicting our choice of $\overline{a} \notin R_1$.

Now, consider the relation $R' = \{a_0 a_1 \dots a_{\ell-k} \mid a_0 0^k a_1 \dots a_{\ell-k} 1^{r-\ell} \in R\}$, which is the result of pinning columns 2 to $(k+1)$ of R to zero and columns $(r-\ell+1)$ to $(r+1)$ to one and discarding the resulting constant columns. R' contains $0^{\ell-k+1}$ and $1^{\ell-k+1}$ but is not complete, since $10^{\ell-k} \notin R'$. By Lemma 4.3, R' and, hence, R 3-simulates equality.

Case 2.2. *Both R_0 and R_1 have width zero,* i.e., are complete relations, possibly padded with constant columns. For $i \in [1,r]$, let R'_i be the relation obtained from R by projecting onto its first and $(i+1)$st columns. Since R has no constant columns, R'_i is either complete, $R_=$, $R_{\neq}$, R_{OR}, R_{NAND}, $R_{\rightarrow}$ or $R_{\leftarrow}$. If there is a k such that R'_k is $R_=$, $R_{\neq}$, $R_{\rightarrow}$ or $R_{\leftarrow}$, then $R_=$, $R_{\neq}$ or $R_{\rightarrow}$ is ppp-definable in R and hence R 3-simulates equality by Lemma 4.2. If there are k_1 and k_2 such that $R'_{k_1} = R_{\mathrm{OR}}$ and $R'_{k_2} = R_{\mathrm{NAND}}$, then R 3-simulates equality by Lemma 4.4. It remains to consider the following two cases.

Case 2.2.1. *Each R'_i is either R_{OR} or complete.* R_1 must be complete, which contradicts the assumption that $R_0 \nsubseteq R_1$.

Case 2.2.1. *Each R'_i is either R_{NAND} or complete.* R_0 must be complete. Let $I = \{i \mid R'_i = R_{\mathrm{NAND}}\}$. Then $R = \bigwedge_{i \in I} \mathrm{NAND}(x_1, x_{i+1})$, so $R \in$ NAND-conj. ∎

Using the duality between OR-conj and NAND-conj relations, we can prove the corresponding result for $R_0, R_1 \in$ NAND-conj. The proof of the classification is completed by a simple induction on the arity of R. Decomposing R as $R_0 + R_1$ and assuming inductively that R_0 and R_1 are of one of the stated types, we use the previous results in this section and Lemma 4.4 to show that R is.

Theorem 5.2. *Every Boolean relation is* OR*-conj or* NAND*-conj or 3-simulates equality.* ∎

6. Complexity

The complexity of approximating #CSP(Γ) where the degree of instances is unbounded is given by Dyer, Goldberg and Jerrum [9, Theorem 3].

Theorem 6.1. *Let Γ be a Boolean constraint language.*

- *If every $R \in \Gamma$ is affine, then* #CSP(Γ) $\in$ **FP**.
- *Otherwise, if $\Gamma \subseteq$ IM-conj, then* #CSP(Γ) $\equiv_{\mathrm{AP}}$ #BIS.
- *Otherwise,* #CSP(Γ) $\equiv_{\mathrm{AP}}$ #SAT.

Working towards our classification of the approximation complexity of #CSP(Γ), we first deal with subcases. The IM-conj case and OR-conj/NAND-conj cases are based on links between those classes of relations and the problems of counting independent sets in bipartite and general graphs, respectively [8, 9], the latter extended to hypergraphs.

Proposition 6.2. *If $\Gamma \subseteq$ IM-conj contains at least one non-affine relation, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{BIS}$ for all $d \geqslant 3$.* ■

Proposition 6.3. *Let R be an* OR*-conj or* NAND*-conj relation of width w. Then, for $d \geqslant 2$, $\#w\text{-HIS}_d \leqslant_{\mathrm{AP}} \#\mathrm{CSP}_d(\{R\} \cup \Gamma_{\mathrm{pin}})$.* ■

Proposition 6.4. *Let R be an* OR*-conj or* NAND*-conj relation of width w. Then, for $d \geqslant 2$, $\#\mathrm{CSP}_d(\{R\} \cup \Gamma_{\mathrm{pin}}) \leqslant_{\mathrm{AP}} \#w\text{-HIS}_{kd}$, where k is the greatest number of times that any variable appears in the normalized formula defining R.* ■

We now give the complexity of approximating $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}})$ for $d \geqslant 3$.

Theorem 6.5. *Let Γ be a Boolean constraint language and let $d \geqslant 3$.*

- *If every $R \in \Gamma$ is affine, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \in$* **FP***.*
- *Otherwise, if $\Gamma \subseteq$ IM-conj, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{BIS}$.*
- *Otherwise, if $\Gamma \subseteq$* OR*-conj or $\Gamma \subseteq$* NAND*-conj, then let w be the greatest width of any relation in Γ and let k be the greatest number of times that any variable appears in the normalized formulae defining the relations of Γ. Then $\#w\text{-HIS}_d \leqslant_{\mathrm{AP}} \#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \leqslant_{\mathrm{AP}} \#w\text{-HIS}_{kd}$.*
- *Otherwise, $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{SAT}$.*

Proof. The affine case is immediate from Theorem 6.1. ($\Gamma \cup \Gamma_{\mathrm{pin}}$ is affine if, and only if, Γ is.) Otherwise, if $\Gamma \subseteq$ IM-conj and some $R \in \Gamma$ is not affine, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{BIS}$ by Proposition 6.2. Otherwise, if $\Gamma \subseteq$ OR-conj or $\Gamma \subseteq$ NAND-conj, then $\#w\text{-HIS}_d \leqslant_{\mathrm{AP}} \#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \leqslant_{\mathrm{AP}} \#w\text{-HIS}_{kd}$ by Propositions 6.3 and 6.4.

Finally, suppose that Γ is not affine, $\Gamma \nsubseteq$ IM-conj, $\Gamma \nsubseteq$ OR-conj and $\Gamma \nsubseteq$ NAND-conj. Since $(\Gamma \cup \Gamma_{\mathrm{pin}})$ is neither affine or a subset of IM-conj, we have $\#\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{SAT}$ by Theorem 6.1 so, if we can show that Γ d-simulates equality, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}})$ by Proposition 4.1 and we are done. If Γ contains a R relation that is neither OR-conj nor NAND-conj, then R 3-simulates equality by Theorem 5.2. Otherwise, Γ must contain distinct relations $R_1 \in$ OR-conj and $R_2 \in$ NAND-conj that are non-affine so have width at least two. So Γ 3-simulates equality by Lemma 4.4. ■

Unless **NP** = **RP**, there is no FPRAS for counting independent sets in graphs of maximum degree at least 25 [7], and, therefore, no FPRAS for $\#w\text{-HIS}_d$ with $r \geqslant 2$ and $d \geqslant 25$. Further, since #SAT is complete for #**P** under AP-reductions [8], #SAT cannot have an FPRAS unless **NP** = **RP**. From Theorem 6.5 above we have the following corollary.

Corollary 6.6. *Let Γ be a Boolean constraint language and let $d \geqslant 25$.*

- *If every $R \in \Gamma$ is affine, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \in$* **FP***.*
- *Otherwise, if $\Gamma \subseteq$ IM-conj, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{BIS}$.*
- *Otherwise there is no FPRAS for $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}})$, unless* **NP** = **RP***.* ■

$\Gamma \cup \Gamma_{\mathrm{pin}}$ is affine (respectively, in OR-conj or in NAND-conj) if, and only if Γ is, so the case for large-degree instances ($d \geqslant 25$) corresponds exactly in complexity to the unbounded case [9]. The case for lower degree bounds is more complex. To put Theorem 6.5 in context, we summarize the known approximability of $\#w\text{-HIS}_d$, parameterized by d and w.

The case $d = 1$ is clearly in **FP** (Theorem 2.1) and so is the case $d = w = 2$, which corresponds to counting independent sets in graphs of maximum degree two. For $d = 2$ and width $w \geqslant 3$, Dyer and Greenhill have shown that there is an FPRAS for $\#w\text{-HIS}_d$ [11]. For $d = 3$, they have shown that there is an FPRAS if the the width w is at most 3.

Degree d	Width w	Approximability of $\#w\text{-HIS}_d$
1	$\geqslant 2$	**FP**
2	2	**FP**
2	$\geqslant 3$	FPRAS [11]
3	$2, 3$	FPRAS [11]
$3, 4, 5$	2	PTAS [30]
$6, \dots, 24$	$\geqslant 2$	The MCMC method is likely to fail [7]
$\geqslant 25$	$\geqslant 2$	No FPRAS unless $\mathbf{NP} = \mathbf{RP}$ [7]

Table 1: Approximability of $\#w\text{-HIS}_d$ (still open for all other values of d and w).

For larger width, the approximability of $\#w\text{-HIS}_3$ is still not known. With the width restricted to $w = 2$ (normal graphs), Weitz has shown that, for degree $d \in \{3, 4, 5\}$, there is a deterministic approximation scheme that runs in polynomial time (a PTAS) [30]. This extends a result of Luby and Vigoda, who gave an FPRAS for $d \leqslant 4$ [24]. For $d > 5$, approximating $\#w\text{-HIS}_d$ becomes considerably harder. More precisely, Dyer, Frieze and Jerrum have shown that for $d = 6$ the Monte Carlo Markov chain technique is likely to fail, in the sense that "cautious" Markov chains are provably slowly mixing [7]. They also showed that, for $d = 25$, there can be no polynomial-time algorithm for approximate counting, unless $\mathbf{NP} = \mathbf{RP}$. These results imply that for $d \in \{6, \dots, 24\}$ and $w \geqslant 2$ the Monte Carlo Markov chain technique is likely to fail and for $d \geqslant 25$ and $w \geqslant 2$, there can be no FPRAS unless $\mathbf{NP} = \mathbf{RP}$. Table 1 summarizes the results.

Returning to bounded-degree #CSP, the case $d = 2$ seems to be rather different to degree bounds three and higher. This is also the case for decision CSP — recall that degree-d $\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}})$ has the same complexity as unbounded-degree $\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}})$ for all $d \geqslant 3$ [6], while degree-2 $\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}})$ is often easier than the unbounded-degree case [6,13] but the complexity of degree-2 $\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}})$ is still open for some Γ.

Our key techniques for determining the complexity of $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}})$ for $d \geqslant 3$ were the 3-simulation of equality and Theorem 5.2, which says that every Boolean relation is in OR-conj, in NAND-conj or 3-simulates equality. However, it seems that not all relations that 3-simulate equality also 2-simulate equality so the corresponding classification of relations does not appear to hold. It seems that different techniques will be required for the degree-2 case. For example, it is possible that there is no FPRAS for $\#\mathrm{CSP}_3(\Gamma \cup \Gamma_{\mathrm{pin}})$ except when Γ is affine. However, Bubley and Dyer have shown that there is an FPRAS for degree-2 #SAT, even though the exact counting problem is $\#\mathbf{P}$-complete [1]. This shows that there is a class $\mathcal{C}$ of constraint languages for which $\#\mathrm{CSP}_2(\Gamma \cup \Gamma_{\mathrm{pin}})$ has an FPRAS for every $\Gamma \in \mathcal{C}$ but for which no exact polynomial-time algorithm is known.

We leave the complexity of degree-2 #CSP and of #BIS and the the various parameterized versions of the counting hypergraph independent sets problem as open questions.

References

[1] R. Bubley and M. Dyer. Graph orientations with no sink and an approximation for a hard case of #SAT. In *8th ACM–SIAM Symp. on Discrete Algorithms (SODA 1997)*, pages 248–257, 1997.

[2] A. A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element domain. *J. ACM*, 53(1):66–120, 2006.

[3] A. A. Bulatov. The complexity of the counting constraint satisfaction problem. In *35th Intl Colloq. on Automata, Languages and Programming (ICALP 2008) Part I*, volume 5125 of *LNCS*, pages 646–661. Springer, 2008.

[4] J.-Y. Cai, P. Lu, and M. Xia. The complexity of complex weighted Boolean #CSP. Upcoming journal submission, 2009.

[5] N. Creignou and M. Hermann. Complexity of generalized satisfiablility counting problems. *Inform. and Comput.*, 125(1):1–12, 1996.

[6] V. Dalmau and D. K. Ford. Generalized satisfiability with limited occurrences per variable: A study through Delta-matroid parity. In *Math. Founds of Comput. Sci. (MFCS 2003)*, volume 2747 of *LNCS*, pages 358–367. Springer, 2003.

[7] M. Dyer, A. Frieze, and M. Jerrum. On counting independent sets in sparse graphs. *SIAM J. Computing*, 31(5):1527–1541, 2002.

[8] M. Dyer, L. A. Goldberg, C. S. Greenhill, and M. Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2003.

[9] M. Dyer, L. A. Goldberg, and M. Jerrum. An approximation trichotomy for Boolean #CSP. To appear in *J. Comput. Sys. Sci.* http://arxiv.org/abs/0710.4272, 2007.

[10] M. Dyer, L. A. Goldberg, and M. Jerrum. The complexity of weighted Boolean CSP. *SIAM J. Comput.*, 38(5):1970–1986, 2009.

[11] M. Dyer and C. S. Greenhill. On Markov chains for independent sets. *J. Algorithms*, 35(1):17–49, 2000.

[12] R. Fagin, L. J. Stockmeyer, and M. Y. Vardi. On monadic NP vs monadic co-NP. *Inform. and Comput.*, 120(1):78–92, 1995.

[13] T. Feder. Fanout limitations on constraint systems. *Theor. Comput. Sci.*, 255(1–2):281–293, 2001.

[14] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

[15] E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *8th Conf. of American Assoc. for Art. Intelligence*, pages 4–9. AAAI Press/MIT Press, 1990.

[16] P. Hell and J. Nešetřil. On the complexity of h-coloring. *J. Combin. Theory B*, 48(1):92–110, 1990.

[17] P. Hell and J. Nešetřil. *Graphs and Homomorphisms.* Oxford University Press, 2004.

[18] M. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[19] D. E. Knuth. The Art of Computer Programming, Vol. 4A: Combinatorial Algorithms. In preparation.

[20] Ph. G. Kolaitis and M. Y. Vardi. Conjunctive query containment and constraint satisfaction. *J. Comput. Sys. Sci.*, 61(2):302–332, 2000.

[21] Ph. G. Kolaitis and M. Y. Vardi. A game-theoretic approach to constraint satisfaction. In *17th Conf. of American Assoc. for Artif. Intelligence*, pages 175–181. AAAI Press/MIT Press, 2000.

[22] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13(1):33–42, 1992.

[23] R. E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975.

[24] M. Luby and E. Vigoda. Fast convergence of the Glauber dynamics for sampling independent sets. *Random Structures and Algorithms*, 15(3–4):229–241, 1999.

[25] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Inform. Sci.*, 7:95–135, 1974.

[26] T. J. Schaefer. The complexity of satisfiability problems. In *10th ACM Symp. on Theory of Computing*, pages 216–226, 1978.

[27] S. Toda. On the computational power of PP and $\bigoplus$P. In *30th Ann. Symp. on Founds of Comput. Sci. (FOCS 1989)*, pages 514–519. IEEE Computer Society, 1989.

[28] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[29] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[30] D. Weitz. Counting independent sets up to the tree threshold. In *38th ACM Symp. on Theory of Computing*, pages 140–149, 2006.

[31] D. Welsh. *Complexity: Knots, Colourings and Counting.* Cambridge University Press, 1993.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 335-346
www.stacs-conf.org

THE COMPLEXITY OF THE LIST HOMOMORPHISM PROBLEM FOR GRAPHS

LÁSZLÓ EGRI [1] AND ANDREI KROKHIN [2] AND BENOIT LAROSE [3] AND PASCAL TESSON [4]

[1] School of Computer Science, McGill University, Montréal, Canada
E-mail address: laszlo.egri@mail.mcgill.ca

[2] School of Engineering and Computing Sciences, Durham University, Durham, UK
E-mail address: andrei.krokhin@durham.ac.uk

[3] Department of Mathematics and Statistics, Concordia University, Montréal, Canada
E-mail address: larose@mathstat.concordia.ca

[4] Department of Computer Science, Laval University, Quebec City, Canada
E-mail address: pascal.tesson@ift.ulaval.ca

Abstract. We completely classify the computational complexity of the list **H**-colouring problem for graphs (with possible loops) in combinatorial and algebraic terms: for every graph **H** the problem is either NP-complete, NL-complete, L-complete or is first-order definable; descriptive complexity equivalents are given as well via Datalog and its fragments. Our algebraic characterisations match important conjectures in the study of constraint satisfaction problems.

1. Introduction

Homomorphisms of graphs, i.e. edge-preserving mappings, generalise graph colourings, and can model a wide variety of combinatorial problems dealing with mappings and assignments [17]. Because of the richness of the homomorphism framework, many computational aspects of graph homomorphisms have recently become the focus of much attention. In the *list* **H**-*colouring* problem (for a fixed graph **H**), one is given a graph **G** and a list L_v of vertices of **H** for each vertex v in **G**, and the goal is to determine whether there is a homomorphism h from **G** to **H** such that $h(v) \in L_v$ for all v. The complexity of such problems has been studied by combinatorial methods, e.g., in [13, 14]. In this paper, we study the complexity of the list homomorphism problem for graphs in the wider context of classifying the complexity of constraint satisfaction problems (CSP), see [3, 15, 18]. It is well known that the CSP can be viewed as the problem of deciding whether there exists a homomorphism from a relational structure to another, thus naturally extending the graph homomorphism problem.

Key words and phrases: graph homomorphism, constraint satisfaction problem, complexity, universal algebra, Datalog.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2467

One line of CSP research studies the non-uniform CSP, in which the target (or template) structure $\mathbf{T}$ is fixed and the question is whether there exists a homomorphism from an input structure to $\mathbf{T}$. Over the last years, much work has been done on classifying the complexity of this problem, denoted $\mathrm{Hom}(\mathbf{T})$ or $\mathrm{CSP}(\mathbf{T})$, with respect to the fixed target structure, see surveys [6, 7, 8, 18]. Classification here is understood with respect to both computational complexity (i.e. membership in a given complexity class such as P, NL, or L, modulo standard assumptions) and descriptive complexity (i.e. definability of the class of all positive, or all negative, instances in a given logic).

The best-known classification results in this direction concern the distinction between polynomial-time solvable and NP-complete CSPs. For example, a classical result of Hell and Nešetřil (see [17, 18]) shows that, for a graph $\mathbf{H}$, $\mathrm{Hom}(\mathbf{H})$ (aka $\mathbf{H}$-colouring) is tractable if $\mathbf{H}$ is bipartite or admits a loop, and is NP-complete otherwise, while Schaefer's dichotomy [24] proves that any Boolean CSP is either in P or NP-complete. Recent work [1] established a more precise classification in the Boolean case: if $\mathbf{T}$ is a structure on $\{0, 1\}$ then $\mathrm{CSP}(\mathbf{T})$ is either NP-complete, P-complete, NL-complete, $\oplus$L-complete, L-complete or in AC^0.

Much of the work concerning the descriptive complexity of CSPs is centred around the database-inspired logic programming language Datalog and its fragments (see [6, 9, 12, 15, 20]). Feder and Vardi initially showed [15] that a number of important tractable cases of $\mathrm{CSP}(\mathbf{T})$ correspond to structures for which $\neg\,\mathrm{CSP}(\mathbf{T})$ (the complement of $\mathrm{CSP}(\mathbf{T})$) is definable in Datalog. Similar ties were uncovered more recently between the two fragments of Datalog known as linear and symmetric Datalog and structures $\mathbf{T}$ for which $\mathrm{CSP}(\mathbf{T})$ belongs to NL and L, respectively [9, 12].

Algebra, logic and combinatorics provide three angles of attack which have fueled progress in this classification effort [6, 7, 8, 17, 18, 20]. The algebraic approach (see [7, 8]) links the complexity of $\mathrm{CSP}(\mathbf{T})$ to the set of functions that preserve the relations in $\mathbf{T}$. In this framework, one associates to each $\mathbf{T}$ an algebra $\mathbb{A}_{\mathbf{T}}$ and exploits the fact that the properties of $\mathbb{A}_{\mathbf{T}}$ completely determine the complexity of $\mathrm{CSP}(\mathbf{T})$. This angle of attack was crucial in establishing key results in the field (see, for example, [2, 5, 7]).

Tame Congruence Theory, a deep universal-algebraic framework first developed by Hobby and McKenzie in the mid 80's [19], classifies the local behaviour of finite algebras into five *types* (unary, affine, Boolean, lattice and semilattice.) It was recently shown (see [6, 7, 22]) that there is a strong connection between the computational and descriptive complexity of $\mathrm{CSP}(\mathbf{T})$ and the set of types that appear in $\mathbb{A}_{\mathbf{T}}$ and its subalgebras. There are strong conditions involving types which are sufficient for NL-hardness, P-hardness and NP-hardness of $\mathrm{CSP}(\mathbf{T})$ as well as for inexpressibility of $\neg\,\mathrm{CSP}(\mathbf{T})$ in Datalog, linear Datalog and symmetric Datalog. These sufficient conditions are also suspected (and in some cases proved) to be necessary, under natural complexity-theoretic assumptions. For example, (a) the presence of unary type is known to imply NP-completeness, while its absence is conjectured to imply tractability (see [7]); (b) the absence of unary and affine types was recently proved to be equivalent to definability in Datalog [2]; (c) the absence of unary, affine, and semilattice types is proved necessary, and suspected to be sufficient, for membership in NL and definability in linear Datalog [22]; (d) the absence of all types but Boolean is proved necessary, and suspected to be sufficient, for membership in L and definability in symmetric Datalog [22]. The strength of evidence varies from case to case and, in particular, the conjectured algebraic conditions concerning CSPs in NL and L (and, as mentioned above, linear and symmetric Datalog) still rest on relatively limited evidence [6, 9, 11, 10, 22].

The aim of the present paper is to show that these algebraic conditions are indeed sufficient and necessary in the special case of list $\mathbf{H}$-colouring for undirected graphs (with possible loops), and to characterise, in this special case, the dividing lines in graph-theoretic terms (both via forbidden subgraphs and through an inductive definition). One can view the list $\mathbf{H}$-colouring problem as a CSP where the template is the structure $\mathbf{H}^L$ consisting of the binary (edge) relation of $\mathbf{H}$ and all unary relations on H (i.e. every subset of H). Tractable list homomorphism problems for general structures were characterised in [5] in algebraic terms. The tractable cases for graphs were described in [14] in both combinatorial and (more specific) algebraic terms; the latter implies, when combined with a recent result [10], that in these cases $\neg\,\mathrm{CSP}(\mathbf{H}^L)$ definable in linear Datalog and therefore $\mathrm{CSP}(\mathbf{H}^L)$ is in fact in NL. We complete the picture by refining this classification and showing that $\mathrm{CSP}(\mathbf{H}^L)$ is either NP-complete, or NL-complete, or L-complete or in AC^0 (and in fact first-order definable). We also remark that the problem of recognising into which case the problem $\mathrm{CSP}(\mathbf{H}^L)$ falls can be solved in polynomial time.

As we mentioned above, the distinction between NP-complete cases and those in NL follows from earlier work [14], and the situation is similar with distinction between L-hard cases and those leading to membership in AC^0 [21, 22]. Therefore, the main body of technical work in the paper concerns the distinction between NL-hardness and membership in L. We give two equivalent characterisations of the class of graphs $\mathbf{H}$ such that $\mathrm{CSP}(\mathbf{H}^L)$ is in L. One characterisation is via forbidden subgraphs (for example, the reflexive graphs in this class are exactly the (P_4, C_4)-free graphs, while the irreflexive ones are exactly the bipartite (P_6, C_6)-free graphs), while the other is via an inductive definition. The first characterisation is used to show that graphs outside of this class give rise to NL-hard problems; we do this by providing constructions witnessing the presence of a non-Boolean type in the algebras associated with the graphs. The second characterisation is used to prove positive results. We first provide operations in the associated algebra which satisfy certain identities; this allows us to show that the necessary condition on types is also sufficient in our case. We also use the inductive definition to demonstrate that the class of negative instances of the corresponding CSP is definable in symmetric Datalog, which implies membership of the CSP in L.

2. Preliminaries

2.1. Graphs and relational structures

In the following we denote the underlying universe of a structure $\mathbf{S}$, $\mathbf{T}$, ... by its roman equivalent S, T, etc. A signature is a (finite) set of relation symbols with associated arities. Let $\mathbf{T}$ be a structure of signature τ; for each relation symbol $R \in \tau$ we denote the corresponding relation of $\mathbf{T}$ by $R(\mathbf{T})$. Let $\mathbf{S}$ be a structure of the same signature. A *homomorphism* from $\mathbf{S}$ to $\mathbf{T}$ is a map f from S to T such that $f(R(\mathbf{S})) \subseteq R(\mathbf{T})$ for each $R \in \tau$. In this case we write $f : \mathbf{S} \to \mathbf{T}$. A structure $\mathbf{T}$ is called a *core* if every homomorphism from $\mathbf{T}$ to itself is a permutation on T. We denote by $\mathrm{CSP}(\mathbf{T})$ the class of all τ-structures $\mathbf{S}$ that admit a homomorphism to $\mathbf{T}$, and by $\neg\,\mathrm{CSP}(\mathbf{T})$ the complement of this class.

The *direct n-th power* of a τ-structure $\mathbf{T}$, denoted $\mathbf{T}^n$, is defined to have universe T^n and, for any (say m-ary) $R \in \tau$, $(\mathbf{a}_1, \ldots, \mathbf{a}_m) \in R(\mathbf{T}^n)$ if and only if $(\mathbf{a}_1[i], \ldots, \mathbf{a}_m[i]) \in R(\mathbf{T})$ for each $1 \leq i \leq n$. For a subset $I \subseteq T$, the *substructure induced by I on* $\mathbf{T}$ is the structure $\mathbf{I}$ with universe I and such that $R(\mathbf{I}) = R(\mathbf{T}) \cap I^m$ for every m-ary $R \in \tau$.

For the purposes of this paper, a *graph* is a relational structure $\mathbf{H} = \langle H; \theta \rangle$ where θ is a symmetric binary relation on H. The graph $\mathbf{H}$ is *reflexive* (*irreflexive*) if $(x, x) \in \theta$ ($(x, x) \notin \theta$) for all $x \in H$. Given a graph $\mathbf{H}$, let $S_1, \ldots, S_k$ denote all subsets of H; let $\mathbf{H}^L$ be the relational structure obtained from $\mathbf{H}$ by adding all the S_i as unary relations; more precisely, let τ be the signature that consists of one binary relational symbol θ and unary symbols R_i, $i = 1, \ldots, k$. The τ-structure $\mathbf{H}^L$ has universe H, $\theta(\mathbf{H}^L)$ is the edge relation of $\mathbf{H}$, and $R_i(\mathbf{H}^L) = S_i$ for all $i = 1, \ldots, k$. It is easy to see that $\mathbf{H}^L$ is a core. We call $\mathrm{CSP}(\mathbf{H}^L)$ the *list homomorphism problem for* $\mathbf{H}$. Note that if $\mathbf{G}$ is an instance of this problem then $\theta(\mathbf{G})$ can be considered as a digraph, but the directions of the arcs are unimportant because $\mathbf{H}$ is undirected. Also, if an element $v \in G$ is in $R_i(\mathbf{G})$ then this is equivalent to v having S_i as its list, so $\mathbf{G}$ can be thought of as a digraph with $\mathbf{H}$-lists.

In [14], a dichotomy result was proved, identifying bi-arc graphs as those whose list homomorphism problem is tractable, and others as giving rise to NP-complete problems. Let C be a circle with two specified points p and q. A bi-arc is a pair of arcs (N, S) such that N contains p but not q and S contains q but not p. A graph $\mathbf{H}$ is a *bi-arc graph* if there is a family of bi-arcs $\{(N_x, S_x) : x \in H\}$ such that, for every $x, y \in H$, the following hold: (i) if x and y are adjacent, then neither N_x intersects S_y nor N_y intersects S_x, and (ii) if x is not adjacent to y then both N_x intersects S_y and N_y intersects S_x.

2.2. Algebra

An n-ary operation on a set A is a map $f : A^n \to A$, a *projection* is an operation of the form $e^i_n(x_1, \ldots, x_n) = x_i$ for some $1 \leq i \leq n$. Given an h-ary relation θ and an n-ary operation f on the same set A, we say that f *preserves* θ or that θ is *invariant* under f if the following holds: given any matrix M of size $h \times n$ whose columns are in θ, applying f to the rows of M will produce an h-tuple in θ.

A *polymorphism* of a structure $\mathbf{T}$ is an operation f that preserves each relation in $\mathbf{T}$; in this case we also say that $\mathbf{T}$ *admits* f. In other words, an n-ary polymorphism of $\mathbf{T}$ is simply a homomorphism from $\mathbf{T}^n$ to $\mathbf{T}$. With any structure $\mathbf{T}$, one associates an algebra $\mathbb{A}_\mathbf{T}$ whose universe is T and whose operations are all polymorphisms of $\mathbf{T}$. Given a graph $\mathbf{H}$, we let $\mathbb{H}$ denote the algebra associated with $\mathbf{H}^L$. An operation on a set is called *conservative* if it preserves all subsets of the set (as unary relations). So, the operations of $\mathbb{H}$ are the conservative polymorphisms of $\mathbf{H}$. Polymorphisms can provide a convenient language when defining classes of graphs. For example, it was shown in [4] that a graph is a bi-arc graph if and only if it admits a conservative majority operation where a *majority* operation is a ternary operation m satisfying the identities $m(x, x, y) = m(x, y, x) = m(y, x, x) = x$.

In order to state some of our results, we will need the notions of a variety and a term operation. Let I be a signature, i.e. a set of operation symbols f each of a fixed arity (we use the term "signature" for both structures and algebras, this will cause no confusion). An *algebra* of signature I is a pair $\mathbb{A} = \langle A; F \rangle$ where A is a non-empty set (the *universe of* $\mathbb{A}$) and $F = \{f^\mathbb{A} : f \in I\}$ is the set of *basic* operations (for each $f \in I$, $f^\mathbb{A}$ is an operation on A of the corresponding arity). The *term operations* of $\mathbb{A}$ are the operations built from the operations in F and projections by using composition. An algebra all of whose (basic or term) operations are conservative is called a *conservative algebra.* A class of similar algebras (i.e. algebras with the same signature) which is closed under formation of homomorphic images, subalgebras and direct products is called a *variety.* The *variety generated* by an algebra $\mathbb{A}$ is denoted by $\mathcal{V}(\mathbb{A})$, and is the smallest variety containing $\mathbb{A}$, i.e. the class of all homomorphic images of subalgebras of powers of $\mathbb{A}$.

Tame Congruence Theory, as developed in [19], is a powerful tool for the analysis of finite algebras. Every finite algebra has a *typeset*, which describes (in a certain specified sense) the local behaviour of the algebra. It contains one or more of the following 5 *types*: (1) the *unary* type, (2) the *affine* type, (3) the *Boolean* type, (4) the *lattice* type and (5) the *semilattice* type. The numbering of the types is fixed, and they are often referred to by their numbers. Simple algebras, i.e. algebras without non-trivial proper homomorphic images, admit a unique type; the prototypical examples are: any 2-element algebra whose basic operations are all unary has type 1. A finite vector space has type 2. The 2-element Boolean algebra has type 3. The 2-element lattice is the 2-element algebra with two binary operations $\langle\{0,1\};\vee,\wedge\rangle$: it has type 4. The 2-element semilattices are the 2-element algebras with a single binary operation $\langle\{0,1\};\wedge\rangle$ and $\langle\{0,1\};\vee\rangle$: they have type 5. The typeset of a variety $\mathcal{V}$, denoted $typ(\mathcal{V})$, is simply the union of typesets of the algebras in it. We will be mostly interested in type-omitting conditions for varieties of the form $\mathcal{V}(\mathbb{A}_{\mathbf{T}})$, and Corollary 3.2 of [25] says that in this case it is enough to consider the typesets of $\mathbb{A}_{\mathbf{T}}$ and its subalgebras.

On the intuitive level, if $\mathbf{T}$ is a core structure then the typeset $typ(\mathcal{V}(\mathbb{A}_{\mathbf{T}}))$ contains crucial information about the kind of relations that $\mathbf{T}$ can or cannot simulate, thus implying lower/upper bounds on the complexity of CSP($\mathbf{T}$). For our purposes here, it will not be necessary to delve further into the technical aspects of types and typesets. We only note that there is a very tight connection between the kind of equations that are satisfied by the algebras in a variety and the types that are *admitted* or *omitted* by a variety, i.e. those types that do or do not appear in the typesets of algebras in the variety [19].

In this paper, we use ternary operations $f_1,\ldots,f_n$ satisfying the following identities:

$$\begin{aligned} x &= f_1(x,y,y) && (2.1)\\ f_i(x,x,y) &= f_{i+1}(x,y,y) \text{ for all } i=1,\ldots n-1 && (2.2)\\ f_n(x,x,y) &= y. && (2.3)\end{aligned}$$

The following lemma contains some type-omitting results that we use in this paper.

Lemma 2.1. [19] *A finite algebra $\mathbb{A}$ has term operations $f_1,\ldots,f_n$, for some $n\geq 1$, satisfying identities (2.1)–(2.3) if and only if the variety $\mathcal{V}(\mathbb{A})$ omits types 1, 4 and 5.*
If a finite algebra $\mathbb{A}$ has a majority term operation then $\mathcal{V}(\mathbb{A})$ omits types 1, 2 and 5.

We remark in passing that operations satisfying identities (2.1)–(2.3) are also known to characterise a certain algebraic (congruence) condition called $(n+1)$-permutability [19].

2.3. Datalog

Datalog is a query and rule language for deductive databases (see [20]). A Datalog program $\mathcal{D}$ over a (relational) signature τ is a finite set of rules of the form $h\leftarrow b_1\wedge\ldots\wedge b_m$ where h and each b_i are atomic formulas $R_j(v_1,...,v_k)$. We say that h is the *head* of the rule and that $b_1\wedge\ldots\wedge b_m$ is its *body.* Relational predicates R_j which appear in the head of some rule of $\mathcal{D}$ are called *intensional database predicates* (*IDB*s) and are not part of the signature τ. All other relational predicates are called *extensional database predicates* (*EDB*s) and are in τ. So, a Datalog program is a recursive specification of IDBs (from EDBs).

A rule of $\mathcal{D}$ is *linear* if its body contains at most one IDB and is *non-recursive* if its body contains only EDBs. A linear but recursive rule is of the form $I_1(\bar{x}) \leftarrow I_2(\bar{y}) \wedge E_1(\bar{z}_1) \wedge \ldots \wedge E_k(\bar{z}_k)$ where I_1, I_2 are IDBs and the E_i are EDBs (note that the variables occurring in $\bar{x}, \bar{y}, \bar{z}_i$ are not necessarily distinct). Each such rule has a *symmetric* $I_2(\bar{y}) \leftarrow I_1(\bar{x}) \wedge E_1(\bar{z}_1) \wedge \ldots \wedge E_k(\bar{z}_k)$. A Datalog program is *non-recursive* if all its rules are non-recursive, *linear* if all its rules are linear and *symmetric* if it is linear and if the symmetric of each recursive rule of $\mathcal{D}$ is also a rule of $\mathcal{D}$.

A Datalog program $\mathcal{D}$ takes a τ-structure $\mathbf{A}$ as input and returns a structure $\mathcal{D}(\mathbf{A})$ over the signature $\tau' = \tau \cup \{I : I \text{ is an IDB in } \mathcal{D}\}$. The relations corresponding to τ are the same as in $\mathbf{A}$, while the new relations are recursively computed by $\mathcal{D}$, with semantics naturally obtained via least fixed-point of monotone operators. We also want to view a Datalog program as being able to accept or reject an input τ-structure and this is achieved by choosing one of the IDBs of $\mathcal{D}$ as the *goal predicate*: the τ-structure $\mathbf{A}$ is *accepted by* $\mathcal{D}$ if the goal predicate is non-empty in $\mathcal{D}(\mathbf{A})$. Thus every Datalog program with a goal predicate defines a class of structures - those that are accepted by the program.

When using Datalog to study CSP($\mathbf{T}$), one usually speaks of the definability of $\neg$ CSP($\mathbf{T}$) in Datalog (i.e. by a Datalog program) or its fragments (because any class definable in Datalog must be closed under extension). Examples of CSPs definable in Datalog and its fragments can be found, e.g., in [6, 12]. As we mentioned before, any problem CSP($\mathbf{T}$) is tractable if its complement is definable in Datalog, and all such structures were recently identified in [2]. Definability of $\neg$ CSP($\mathbf{T}$) in linear (symmetric) Datalog implies that CSP($\mathbf{T}$) belongs to NL and L, respectively [9, 12]. As we discussed in Section 1, there is a connection between definability of CSPs in Datalog (and its fragments) and the presence/absence of types in the corresponding algebra (or variety).

Note that it follows from Lemma 2.1 and from the results in [22, 26] that if, for a core structure $\mathbf{T}$, $\neg$ CSP($\mathbf{T}$) is definable in symmetric Datalog then $\mathbf{T}$ must admit, for some n, operations satisfying identities (2.1)–(2.3). Moreover, with the result of [2], a conjecture from [22] can be restated as follows: for a core structure $\mathbf{T}$, if $\neg$ CSP($\mathbf{T}$) is definable in Datalog and, for some n, $\mathbf{T}$ admits operations satisfying (2.1)–(2.3), then $\neg$ CSP($\mathbf{T}$) is definable in symmetric Datalog. This conjecture is proved in [11] for $n = 1$.

3. A class of graphs

In this section, we give combinatorial characterisations of a class of graphs whose list homomorphism problem will turn out to belong to L.

Let $\mathbf{H}_1$ and $\mathbf{H}_2$ be bipartite irreflexive graphs, with colour classes B_1, T_1 and B_2 and T_2 respectively, with T_1 and B_2 non-empty. We define the *special sum* $\mathbf{H}_1 \odot \mathbf{H}_2$ (which depends on the choice of the B_i and T_i) as follows: it is the graph obtained from the disjoint union of $\mathbf{H}_1$ and $\mathbf{H}_2$ by adding all possible edges between the vertices in T_1 and B_2. Notice that we can often decompose a bipartite graph in several ways, and even choose B_1 or T_2 to be empty. We say that an irreflexive graph $\mathbf{H}$ is *a special sum* or *expressed as a special sum* if there exist two bipartite graphs and a choice of colour classes on each such that $\mathbf{H}$ is isomorphic to the special sum of these two graphs.

Definition 3.1. Let $\mathcal{K}$ denote the smallest class of irreflexive graphs containing the one-element graph and closed under (i) special sum and (ii) disjoint union. We call the graphs in $\mathcal{K}$ *basic irreflexive*.

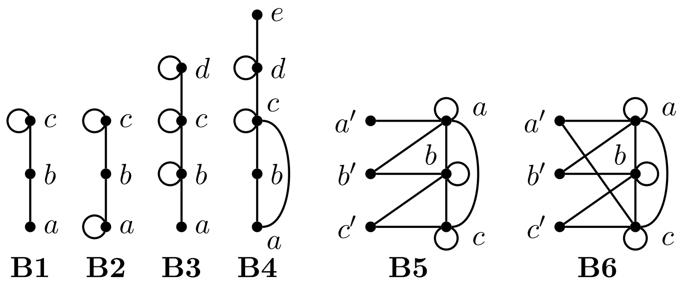


Figure 1: The forbidden mixed graphs.

The following result gives a characterisation of basic irreflexive graphs in terms of forbidden subgraphs:

Lemma 3.2. *Let* **H** *be an irreflexive graph. Then the following conditions are equivalent:*

(1) **H** *is basic irreflexive;*
(2) **H** *is bipartite, contains no induced 6-cycle, nor any induced path of length 5.*

We shall now describe our main family of graphs, first by forbidden induced subgraphs, and then in an inductive manner.

Definition 3.3. Define the class $\mathcal{L}$ of graphs as follows: a graph **H** belongs to $\mathcal{L}$ if it contains none of the following as an induced subgraph:

(1) the reflexive path of length 3 and the reflexive 4-cycle;
(2) the irreflexive cycles of length 3, 5 and 6, and the irreflexive path of length 5;
(3) **B1**, **B2**, **B3**, **B4**, **B5** and **B6** (see Figure 1.)

We will now characterise the class $\mathcal{L}$ in an inductive manner.

Definition 3.4. A connected graph **H** is *basic* if either (i) **H** is a single loop, or (ii) **H** is a basic irreflexive graph, or (iii) **H** is obtained from a basic irreflexive graph $\mathbf{H}_1$ with colour classes B and T by adding every edge (including loops) of the form $\{t, t'\}$ where $t, t' \in T$.

Definition 3.5. Given two vertex-disjoint graphs $\mathbf{H}_1$ and $\mathbf{H}_2$, the *adjunction of* $\mathbf{H}_1$ *to* $\mathbf{H}_2$ is the graph $\mathbf{H}_1 \oslash \mathbf{H}_2$ obtained by taking the disjoint union of the two graphs, and adding every edge of the form $\{x, y\}$ where x is a loop in $\mathbf{H}_1$ and y is a vertex of $\mathbf{H}_2$.

Lemma 3.6. *Let* $\mathcal{L}_R$ *denote the class of reflexive graphs in* $\mathcal{L}$. *Then* $\mathcal{L}_R$ *is the smallest class* $\mathcal{D}$ *of reflexive graphs such that:*

(1) $\mathcal{D}$ *contains the one-element graph;*
(2) $\mathcal{D}$ *is closed under disjoint union;*
(3) *if* $\mathbf{H}_1$ *is a single loop and* $\mathbf{H}_2 \in \mathcal{D}$ *then* $\mathbf{H}_1 \oslash \mathbf{H}_2 \in \mathcal{D}$.

Lemma 3.6 states that the reflexive graphs avoiding the path of length 3 and the 4-cycle are precisely those constructed from the one-element loop using disjoint union and adjunction of a universal vertex. These graphs can also be described by the following property: every connected induced subgraph of size at most 4 has a universal vertex. These graphs have been studied previously as those with NLCT width 1, which were proved to be exactly the trivially perfect graphs [16]. Our result provides an alternative proof of the equivalence of these conditions.

Theorem 3.7. *The class* $\mathcal{L}$ *is the smallest class* $\mathcal{C}$ *of graphs such that:*

(1) *$\mathcal{C}$ contains the basic graphs;*
(2) *$\mathcal{C}$ is closed under disjoint union;*
(3) *if $\mathbf{H}_1$ is a basic graph and $\mathbf{H}_2 \in \mathcal{C}$ then $\mathbf{H}_1 \oslash \mathbf{H}_2 \in \mathcal{C}$.*

Proof. We start by showing that every basic graph is in $\mathcal{L}$, i.e. that a basic graph does not contain any of the forbidden graphs. If $\mathbf{H}$ is a single loop or a basic irreflexive graph, then this is immediate. Otherwise $\mathbf{H}$ is obtained from a basic irreflexive graph $\mathbf{H}_1$ with colour classes B and T by adding every edge of the form (t_1, t_2) where $t_i \in T$. In particular, the loops form a clique and no edge connects two non-loops; it is clear in that case that $\mathbf{H}$ contains none of **B1**, **B2**, **B3**, **B4**. On the other hand if $\mathbf{H}$ contains **B5** or **B6**, then $\mathbf{H}_1$ contains the path of length 5 or the 6-cycle, contradicting the fact that $\mathbf{H}_1$ is basic.

Next we show that $\mathcal{L}$ is closed under disjoint union and adjunction of basic graphs. It is obvious that the disjoint union of graphs that avoid the forbidden graphs will also avoid these. So suppose that an adjunction $\mathbf{H}_1 \oslash \mathbf{H}_2$, where $\mathbf{H}_1$ is a basic graph, contains an induced forbidden graph $\mathbf{B}$ whose vertices are neither all in H_1 nor H_2; without loss of generality H_1 contains at least one loop, its loops form a clique and none of its edges connects two non-loops. It is then easy to verify that $\mathbf{B}$ contains both loops and non-loops. Because the other cases are similar, we prove only that $\mathbf{B}$ is not **B3**: since vertex d is not adjacent to a it must be in $\mathbf{H}_2$, and similarly for c. Since b is not adjacent to d it must also be in $\mathbf{H}_2$; since non-loops of $\mathbf{H}_1$ are not adjacent to elements of $\mathbf{H}_2$ it follows that a is in $\mathbf{H}_2$ also, a contradiction.

Now we must show that every graph in $\mathcal{L}$ can be obtained from the basic graphs by disjoint union and adjunction of basic graphs. Suppose this is not the case. If $\mathbf{H}$ is a counterexample of minimum size, then obviously it is connected, and it contains at least one loop for otherwise it is a basic irreflexive graph. By Lemma 3.6, $\mathbf{H}$ also contains at least one non-loop.

For $a \in H$ let $N(a)$ denote its set of neighbours. Let $\mathbf{R}(\mathbf{H})$ denote the subgraph of $\mathbf{H}$ induced by its set $R(H)$ of loops, and let $\mathbf{J}(\mathbf{H})$ denote the subgraph induced by $J(H)$, the set of non-loops of $\mathbf{H}$. Since $\mathbf{H}$ is connected and neither **B1** nor **B2** is an induced subgraph of $\mathbf{H}$, the graph $\mathbf{R}(\mathbf{H})$ is also connected, and furthermore every vertex in $J(H)$ is adjacent to some vertex in $R(H)$. By Lemma 3.6, we know that $\mathbf{R}(\mathbf{H})$ contains at least one universal vertex: let U denote the (non-empty) set of universal vertices of $\mathbf{R}(\mathbf{H})$. Let J denote the set of all $a \in J(H)$ such that $N(a) \cap R(H) \subseteq U$. Let us show that $J \neq \emptyset$. For every $u \in U$, there is $w \in J(H)$ not adjacent to u because otherwise $\mathbf{H}$ is obtained by adjoining u to the rest of $\mathbf{H}$, a contradiction with the choice of $\mathbf{H}$. If this w has a neighbour $r \in R(H) \setminus U$ then there is some $s \in R(H) \setminus U$ not adjacent to r, and the graph induced by $\{w, u, s, r\}$ contains **B2** or **B3**, a contradiction. Hence, $w \in J$. Let $\mathbf{S}$ denote the subgraph of $\mathbf{H}$ induced by $U \cup J$. The graph $\mathbf{S}$ is connected. We claim that the following properties also hold:

(1) if a and b are adjacent non-loops, then $N(a) \cap U = N(b) \cap U$;
(2) if a is in a connected component of the subgraph of $\mathbf{S}$ induced by J with more than one vertex, then for any other $b \in J$, one of $N(a) \cap U, N(b) \cap U$ contains the other.

The first statement holds because **B1** is forbidden, and the second follows from the first because **B4** is also forbidden. Let $J_1, \ldots, J_k$ denote the different connected components of J in $\mathbf{S}$. By (1) we may let $N(J_i)$ denote the set of common neighbours of members of J_i in U. By (2), we can re-order the J_i's so that for some $1 \leq m \leq k$ we have $N(J_i) \subseteq N(J_j)$ for all $i \leq m$ and all $j > m$, and, in addition, we have $m = 1$ or $|J_i| = 1$ for all $1 \leq i \leq m$. Let $\mathbf{B}$ denote the subgraph of $\mathbf{S}$ induced by $B = \bigcup_{i=1}^{m} (J_i \cup N(J_i))$, and let $\mathbf{C}$ be the subgraph of $\mathbf{H}$ induced by $H \setminus B$. We claim that $\mathbf{H} = \mathbf{B} \oslash \mathbf{C}$. For this, it suffices to show that every element in $\bigcup_{i=1}^{m} N(J_i)$ is adjacent to every non-loop $c \in C$. By construction this holds if $c \in J \cap C$. Now suppose this does not hold: then some $x \in J(H) \setminus J$ is not adjacent to some $y \in N(J_i)$ for some $i \leq m$. Since $x \notin J$ we may find some $z \in R(H) \setminus U$ adjacent to x; it is of course also adjacent to y. Since $z \notin U$ there exists some $z' \in R(H) \setminus U$ that is not adjacent to z, but it is of course adjacent to y. If x is adjacent to z', then $\{x, z, z'\}$ induces a subgraph isomorphic to **B2**, a contradiction. Otherwise, $\{x, z, y, z'\}$ induces a subgraph isomorphic to **B3**, also a contradiction.

If every J_i with $i \leq m$ contains a single element, notice that $\mathbf{B}$ is a basic graph: indeed, removing all edges between its loops yields a bipartite irreflexive graph which contains neither the path of length 5 nor the 6-cycle, since $\mathbf{B}$ contains neither **B5** nor **B6**. Since this contradicts our hypothesis on $\mathbf{H}$, we conclude that $m = 1$. But this means that $N(J_1)$ is a set of universal vertices in $\mathbf{H}$. Let u be such a vertex and let D denote its complement in $\mathbf{H}$: clearly $\mathbf{H}$ is obtained as the adjunction of the single loop u to D, contradicting our hypothesis. This concludes the proof. ∎

4. Classification results

Recall the standard numbering of types: (1) *unary*, (2) *affine* , (3) *Boolean*, (4) *lattice* and (5) *semilattice*. We will need the following auxiliary result (which is well known). Note that the assumptions of this lemma effectively say that $\mathrm{CSP}(\mathbf{T})$ can simulate the graph k-colouring problem (with $k = |U|$) or the directed st-connectivity problem.

Lemma 4.1. *Let* $\mathbf{S}, \mathbf{T}$ *be structures, let* $s_1, s_2 \in S$, *and let* $R = \{(f(s_1), f(s_2)) \mid f : \mathbf{S} \to \mathbf{T}\}$.

1. *If* $R = \{(x, y) \in U^2 \mid x \neq y\}$ *for some subset* $U \subseteq T$ *with* $|U| \geq 3$ *then* $\mathcal{V}(\mathbb{A}_{\mathbf{T}})$ *admits type 1.*
2. *If* $R = \{(t, t), (t, t'), (t', t')\}$ *for some distinct* $t, t' \in T$ *then* $\mathcal{V}(\mathbb{A}_{\mathbf{T}})$ *admits at least one of the types 1, 4, 5.*

Proof [sketch]: The assumption of this lemma implies that $\mathbb{A}_{\mathbf{T}}$ has a subalgebra (induced by U and $\{t, t'\}$, respectively) such that all operations of the subalgebra preserve the relation R. It is well-known (see, e.g., [17]) that all operations preserving the disequality relation on U are essentially unary, while it is easy to check that the order relation on a 2-element set cannot admit operations satisfying identities (2.1)–(2.3), so one can use Lemma 2.1. ∎

The following lemma connects the characterisation of bi-arc graphs given in [4] with a type-omitting condition.

Lemma 4.2. *Let* $\mathbf{H}$ *be a graph. Then the following conditions are equivalent:*

1. *the variety* $\mathcal{V}(\mathbb{H})$ *omits type 1;*
2. *the graph* $\mathbf{H}$ *admits a conservative majority operation;*
3. *the graph* $\mathbf{H}$ *is a bi-arc graph.*

The results summarised in the following theorem are known (or easily follow from known results, with a little help from Lemma 4.2).

Theorem 4.3. *Let* $\mathbf{H}$ *be a graph.*

- *If typ*$(\mathcal{V}(\mathbb{H}))$ *admits type 1, then* $\neg\,\mathrm{CSP}(\mathbf{H}^L)$ *is not expressible in Datalog and* $\mathrm{CSP}(\mathbf{H}^L)$ *is* NP*-complete (under first-order reductions);*

- *if* $typ(\mathcal{V}(\mathbb{H}))$ *omits type 1 but admits type 4 then* $\neg\,\mathrm{CSP}(\mathbf{H}^L)$ *is not expressible in symmetric Datalog but is expressible in linear Datalog, and* $\mathrm{CSP}(\mathbf{H}^L)$ *is* NL-*complete (under first-order reductions.)*

Proof. The first statement is shown in [22]. If the variety omits type 1, then $\mathbf{H}^L$ admits a majority operation by Lemma 4.2 and then $\neg\,\mathrm{CSP}(\mathbf{H}^L)$ is expressible in linear Datalog by [10]; in particular the problem is in NL. If, furthermore, the variety admits type 4, then $\neg\,\mathrm{CSP}(\mathbf{H}^L)$ is not expressible in symmetric Datalog and is NL-hard by results in [22]. ■

By Lemma 2.1, the presence of a majority operation in $\mathbb{H}$ implies that $typ(\mathcal{V}(\mathbb{H}))$ can contain only types 3 and 4. Type 4 is dealt with in Theorem 4.3, so it remains to investigate graphs $\mathbf{H}$ with $typ(\mathcal{V}(\mathbb{H})) = \{3\}$.

The next theorem is the main result of this paper.

Theorem 4.4. *Let* $\mathbf{H}$ *be a graph. Then the following conditions are equivalent:*

(1) $\mathbf{H}$ *admits conservative operations satisfying (2.1)–(2.3) for* $n = 3$*;*
(2) $\mathbf{H}$ *admits conservative operations satisfying (2.1)–(2.3) for some* $n \geq 1$*;*
(3) $typ(\mathcal{V}(\mathbb{H})) = \{3\}$*;*
(4) $\mathbf{H} \in \mathcal{L}$*;*
(5) $\neg\,\mathrm{CSP}(\mathbf{H}^L)$ *is definable in symmetric Datalog.*

If the above holds then $\mathrm{CSP}(\mathbf{H}^L)$ *is in the complexity class* L.

Proof [sketch]: (1) ⇒ (2) is trivial. If (2) holds then by Lemma 2.1 $\mathcal{V}(\mathbb{H})$ omits types 1, 4, and 5. By Lemma 4.2, $\mathbf{H}$ admits a majority operation, so Lemma 2.1 implies that $\mathcal{V}(\mathbb{H})$ also omits type 2; hence (3) holds. Implication (3)⇒(4) is the content of Lemma 4.5 below, and (5) implies (3) by a result of [22]. By using Theorem 3.7, one can show that (4) implies both (1) and (5). Finally, definability in symmetric Datalog implies membership in L by [12]. ■

Lemma 4.5. *If* $\mathbf{H} \notin \mathcal{L}$ *then* $typ(\mathcal{V}(\mathbb{H})) \neq \{3\}$.

Proof. By Theorem 9.15 of [19], $typ(\mathcal{V}(\mathbb{H})) = \{3\}$ if and only if $\mathbf{H}$ admits a sequence of conservative operations satisfying certain identities (in the spirit of (2.1)–(2.3)). By conservativity, such operations can be restricted to any subset of H while satisfying the same identities, so the property $typ(\mathcal{V}(\mathbb{H})) = \{3\}$ is inherited by induced subgraphs. It follows that it is enough to prove this lemma for the forbidden graphs from Definition 3.3.

For the irreflexive odd cycles, the lemma follows immediately from the main results of [3, 23]. The proof of Theorem 3.1 of [13] shows that the conditions of Lemma 4.1(1) are satisfied by (some $\mathbf{S}, s_1, s_2$ and) $\mathbf{T} = \mathbf{F}^L$ where $\mathbf{F}$ is the irreflexive 6-cycle. One can check that the reflexive 4-cycle is not a bi-arc graph, so we can apply Lemma 4.2 in this case.

For the remaining forbidden graphs $\mathbf{F}$ from Definition 3.3, we use Lemma 4.1(2) with $\mathbf{T} = \mathbf{F}^L$. In each case, the binary relation of the structure $\mathbf{S}$ will be a short undirected path, and s_1, s_2 will be the endpoints of the path. We will represent such a structure $\mathbf{S}$ by a sequence of subsets of F (indicating lists assigned to vertices of the path). It can be easily checked that, in each case, the relation R defined as in Lemma 4.1 is of the required form.

If $\mathbf{F}$ is the reflexive path of length 3, say $a - b - c - d$, then $\mathbf{S} = ac - bc - ad - ac$. If $\mathbf{F}$ is the irreflexive path of length 5, say $a - b - c - d - e - f$ then $\mathbf{S} = ae - bd - ce - bf - ae$. For graphs $\mathbf{B1} - \mathbf{B6}$, we use notation from Fig. 1. For $\mathbf{B1}$, $\mathbf{S} = bc - bc - ab - ab - bc$. For $\mathbf{B2}$, $\mathbf{S} = bc - ac - ab - bc$. For $\mathbf{B3}$, $\mathbf{S} = bc - ad - bd - bc$. For $\mathbf{B4}$, $\mathbf{S} = ae - bd - cd - ae$. Finally, for both $\mathbf{B5}$ and $\mathbf{B6}$, $\mathbf{S} = ac - b'c' - ab - a'c' - ac$. ■

For completeness' sake, we describe graphs whose list homomorphism problem is definable in first-order logic (equivalently, is in AC^0, see [6].) By results in [22], any problem CSP($\mathbf{T}$) is either first-order definable or L-hard under FO reductions. Hence, it follows from Theorem 4.4 that, for a graph $\mathbf{H} \in \mathcal{L}$, the list homomorphism problem for $\mathbf{H}$ is either first-order definable or L-complete.

We need the following characterisation of structures whose CSP is first-order definable [21]. Let $\mathbf{T}$ be a relational structure and let $a, b \in T$. We say that b *dominates* a in $\mathbf{T}$ if for any relation R of $\mathbf{T}$, and any tuple $\overline{t} \in R$, replacement of any occurrence of a by b in $\overline{t}$ will yield a tuple of R. Recall the definition of a direct power of a structure from Subsection 2.1. If $\mathbf{T}$ is a relational structure, we say that the structure $\mathbf{T}^2$ *dismantles to the diagonal* if there exists a sequence of elements $\{a_0, \dots, a_n\} = T^2 \setminus \{(a,a) : a \in T\}$ such that, for all $0 \leq i \leq n$, a_i is dominated in $\mathbf{T}_i$, where $\mathbf{T}_0 = \mathbf{T}^2$ and $\mathbf{T}_i$ is the substructure of $\mathbf{T}^2$ induced by $T^2 \setminus \{a_0, \dots, a_{i-1}\}$ for $i > 0$.

Lemma 4.6 ([21]). *Let* $\mathbf{T}$ *be a core relational structure. Then* CSP($\mathbf{T}$) *is first-order definable if and only if* $\mathbf{T}^2$ *dismantles to the diagonal.*

Theorem 4.7. *Let* $\mathbf{H}$ *be a graph. Then* CSP($\mathbf{H}^L$) *is first-order definable if and only if* $\mathbf{H}$ *has the following form: H is the disjoint union of two sets L and N such that (i) L is the set of loops of* $\mathbf{H}$ *and induces a complete graph, (ii) N is the set of non-loops of* $\mathbf{H}$ *and induces a graph with no edges, and (iii) $N = \{x_1, \dots, x_m\}$ can be ordered so that the neighbourhood of x_i is contained in the neighbourhood of x_{i+1} for all $1 \leq i \leq m-1$.*

Proof. We first prove that conditions (i) and (ii) are necessary. Notice that if CSP($\mathbf{H}^L$) is first-order definable then so is CSP($\mathbf{K}^L$) for any induced substructure $\mathbf{K}$ of $\mathbf{H}$. Let x and y be distinct vertices of $\mathbf{H}$ and let $\mathbf{K}^L$ be the substructure of $\mathbf{H}^L$ induced by $\{x, y\}$. If x and y are non-adjacent loops, then $\theta(\mathbf{K}) = \{(x,x),(y,y)\}$ the equality relation on $\{x,y\}$; if x and y are adjacent non-loops, then $\theta(\mathbf{K}) = \{(x,y),(y,x)\}$, the adjacency relation of the complete graph on 2 vertices. It is well known (and can be easily derived from Lemma 4.6) that neither of these classes CSP($\mathbf{K}^L$) is first-order definable. It follows that the loops of $\mathbf{H}$ induce a complete graph and the non-loops induce a graph with no edges.

Now we prove (iii) is necessary. Suppose for a contradiction that there exist distinct elements x and y of N and elements n and m of L such that m is adjacent to x but not to y, and n is adjacent to y but not to x. Then CSP($\mathbf{G}$) is first-order definable, where $\mathbf{G}$ is the substructure of $\mathbf{H}^L$ induced by $\{x, y, m, n\}$. By Lemma 4.6, $\mathbf{G}^2$ dismantles to the diagonal. Then (x,y) must be dominated by one of (x,x), (y,x) or (y,y), since domination respects the unary relation $\{x,y\}^2$ (on G^2). But (m,n) is a neighbour of (x,y) and none of the other three, a contradiction.

For the converse: we show that we can dismantle $(\mathbf{H}^L)^2$ to the diagonal. Let $x \in H$: then (x_1, x) and (x, x_1) are dominated by (x,x). Suppose that we have dismantled every element containing a coordinate equal to x_i with $i \leq j-1$: if x is any element of H such that the elements (x_j, x) and (x, x_j) remain, then either x is a loop or $x = x_k$ with $k \geq j$; in any case the elements (x_j, x_k) and (x_k, x_j) are dominated by (x,x). In this way we can remove all pairs (x,y) with one of x or y a non-loop. For the remaining pairs, notice that if u and v are any loops then (u,v) is dominated (in what remains of $(\mathbf{H}^L)^2$) by (u,u). ■

Finally, given a graph $\mathbf{H}$, it can be decided in polynomial time which of the different cases delineated in Theorems 4.3, 4.4, 4.7 the list homomorphism problem for $\mathbf{H}$ satisfies. Indeed, it is known that bi-arc graphs can be recognised in polynomial time (see [14]). Assume that $\mathbf{H}$ is a bi-arc graph: the forbidden substructure definition of the class $\mathcal{L}$ gives an AC^0 algorithm to recognise them; and those graphs whose list homomorphism problem is first-order definable can be recognised in polynomial time by results of [21].

References

[1] E. Allender, M. Bauland, N. Immerman, H. Schnoor, and H. Vollmer. The complexity of satisfiability problems: Refining Schaefer's theorem. *Journal of Computer and System Sciences*, 75(4):245–254, 2009.

[2] L. Barto and M. Kozik. Constraint satisfaction probllems of bounded width. In *FOCS'09*, 2009.

[3] L. Barto, M. Kozik, and T. Niven. The CSP dichotomy holds for digraphs with no sources and no sinks (A positive answer to a conjecture of Bang-Jensen and Hell). *SIAM J. Comput.*, 38(5):1782–1802, 2009.

[4] R. Brewster, T. Feder, P. Hell, J. Huang, and G. MacGillavray. Near-unanimity functions and varieties of reflexive graphs. *SIAM J. Discrete Math.*, 22:938–960, 2008.

[5] A. Bulatov. Tractable conservative constraint satisfaction problems. In *LICS'03*, pages 321–330, 2003.

[6] A. Bulatov, A. Krokhin, and B. Larose. Dualities for constraint satisfaction problems. In *Complexity of Constraints*, volume 5250 of *LNCS*, pages 93–124. 2008.

[7] A. Bulatov and M. Valeriote. Recent results on the algebraic approach to the CSP. In *Complexity of Constraints*, volume 5250 of *LNCS*, pages 68–92. 2008.

[8] D. Cohen and P. Jeavons. The complexity of constraint languages. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 8. Elsevier, 2006.

[9] V. Dalmau. Linear Datalog and bounded path duality for relational structures. *Logical Methods in Computer Science*, 1(1), 2005. (electronic).

[10] V. Dalmau and A. Krokhin. Majority constraints have bounded pathwidth duality. *European Journal of Combinatorics*, 29(4):821–837, 2008.

[11] V. Dalmau and B. Larose. Maltsev + Datalog ⇒ Symmetric Datalog. In *LICS'08*, pages 297–306, 2008.

[12] L. Egri, B. Larose, and P. Tesson. Symmetric Datalog and constraint satisfaction problems in Logspace. In *LICS'07*, pages 193–202, 2007.

[13] T. Feder, P. Hell, and J. Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19:487–505, 1999.

[14] T. Feder, P. Hell, and J. Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42:61–80, 2003.

[15] T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Comput.*, 28:57–104, 1998.

[16] F. Gurski. Characterizations of co-graphs defined by restricted NLC-width or clique-width operations. *Discrete Mathematics*, 306(2):271–277, 2006.

[17] P. Hell and J. Nešetřil. *Graphs and Homomorphisms.* Oxford University Press, 2004.

[18] P. Hell and J. Nešetřil. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, 2(3):143–163, 2008.

[19] D. Hobby and R.N. McKenzie. *The Structure of Finite Algebras.* AMS, Providence, R.I., 1988.

[20] Ph.G. Kolaitis and M.Y. Vardi. A logical approach to constraint satisfaction. In *Complexity of Constraints*, volume 5250 of *LNCS*, pages 125–155. 2008.

[21] B. Larose, C. Loten, and C. Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, 3(4), 2007. (electronic).

[22] B. Larose and P. Tesson. Universal algebra and hardness results for constraint satisfaction problems. *Theoretical Computer Science*, 410(18):1629–1647, 2009.

[23] M. Maróti and R. McKenzie. Existence theorems for weakly symmetric operations. *Algebra Univ.*, 59(3-4):463–489, 2008.

[24] T.J. Schaefer. The complexity of satisfiability problems. In *STOC'78*, pages 216–226, 1978.

[25] M. Valeriote. A subalgebra intersection property for congruence-distributive varieties. *Canadian Journal of Mathematics*, 61(2):451–464, 2009.

[26] László Zádori and Benoit Larose. Bounded width problems and algebras. *Algebra Univ.*, 56(3-4):439–466, 2007.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 347-358
www.stacs-conf.org

IMPROVED APPROXIMATION GUARANTEES FOR WEIGHTED MATCHING IN THE SEMI-STREAMING MODEL

LEAH EPSTEIN [1] AND ASAF LEVIN [2] AND JULIÁN MESTRE [3] AND DANNY SEGEV [4]

[1] Department of Mathematics, University of Haifa, 31905 Haifa, Israel.

[2] Chaya fellow. Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel.

[3] Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany.

[4] Department of Statistics, University of Haifa, 31905 Haifa, Israel.

Abstract. We study the maximum weight matching problem in the semi-streaming model, and improve on the currently best one-pass algorithm due to Zelke (Proc. STACS '08, pages 669–680) by devising a deterministic approach whose performance guarantee is $4.91 + \varepsilon$. In addition, we study *preemptive* online algorithms, a sub-class of one-pass algorithms where we are only allowed to maintain a feasible matching in memory at any point in time. All known results prior to Zelke's belong to this sub-class. We provide a lower bound of 4.967 on the competitive ratio of any such deterministic algorithm, and hence show that future improvements will have to store in memory a set of edges which is not necessarily a feasible matching. We conclude by presenting an empirical study, conducted in order to compare the practical performance of our approach to that of previously suggested algorithms.

1. Introduction

The computational task of detecting maximum weight matchings is one of the most fundamental problems in discrete optimization, attracting plenty of attention from the operations research, computer science, and mathematics communities. (For a wealth of references on matching problems see [16].) In such settings, we are given an undirected graph $G = (V, E)$ whose edges are associated with non-negative weights specified by $w : E \to \mathbb{R}_+$. A set of edges $M \subseteq E$ is a *matching* if no two of the edges share a common vertex, that is, the degree of any vertex in (V, M) is at most 1. The weight $w(M)$ of a matching M is defined as the combined weight of its edges, i.e., $\sum_{e \in M} w(e)$. The objective is to compute a matching of maximum weight. We study this problem in two related computational models: the *semi-streaming* model and the *preemptive online* model.

The semi-streaming model. Even though these settings appear to be rather simple as first glance, it is worth noting that matching problems have an abundance of flavors, usually depending on how the input is specified. In this paper, we investigate weighted matchings in the *semi-streaming* model, first suggested by Muthukrishnan [14]. Specifically, a *graph stream* is a sequence $e_{i_1}, e_{i_2}, \ldots$ of

The third author was supported by an Alexander von Humboldt Fellowship.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2476

distinct edges, where $e_{i_1}, e_{i_2}, \ldots$ is an arbitrary permutation of E. When an algorithm is processing the stream, edges are revealed sequentially, one at a time. Letting $n = |V|$ and $m = |E|$, efficiency in this model is measured by the space $S(n, m)$ a graph algorithm uses, the time $T(n, m)$ it requires to process each edge, and the number of passes $P(n, m)$ it makes over the input stream. Throughout the paper, however, we focus on one-pass algorithms, that is, $P(n, m) = 1$. The main restriction is that the space $S(n, m)$ is limited to $O(n \cdot \text{polylog}(n))$ bits of memory. We refer the reader to a number of recent papers [14, 4, 5, 2, 12] and to the references therein for a detailed literature review.

Online version. Online matching has previously been modeled as follows [9]. Edges are presented one by one to the algorithm, along with their weight. Once an edge is presented, we must make an irrevocable decision, whether to accept it or not. An edge may be accepted only if its addition to the set of previously accepted edges forms a feasible matching. In other words, an algorithm must keep a matching at all times, and its final output consists of all edges which were ever accepted. In this model, it is easy to verify that the competitive ratio of any (deterministic or randomized) algorithm exceeds any function of the number of vertices, meaning that no competitive algorithm exists. However, if all weights are equal, a greedy approach which accepts an edge whenever possible, has a competitive ratio of 2, which is best possible for deterministic algorithms [9].

Similarly to other online settings (such as call control problems [6]), a preemptive model can be defined, allowing us to remove a previously accepted edge from the current matching at any point in time; this event is called *preemption*. Nevertheless, an edge which was either rejected or preempted cannot be inserted to the matching later on. We point out that other types of online matching problems were studied as well [9, 7, 10, 1].

Comparison between the models. Both one-pass semi-streaming algorithms and online algorithms perform a single scan of the input. However, unlike semi-streaming algorithms, online algorithms are allowed to concurrently utilize memory for two different purposes. The first purpose is obviously to maintain the current solution, which must always be a feasible matching, implying that the memory size of this nature is bounded by the maximal size of a matching. The second purpose is to keep track of arbitrary information regarding the past, without any concrete bound on the size of memory used. Therefore, in theory, online algorithms are allowed to use much larger memory than is allowed in the semi-streaming model. Moreover, although this possibility is rarely used, online algorithms may perform exponential time computations whenever a new piece of input is revealed. On the other hand, a semi-streaming algorithm may re-insert an edge to the current solution, even if it has been temporarily removed, as long as this edge was kept in memory. This extra power is not allowed for online (preemptive) algorithms, making them inferior in this sense in comparison to their semi-streaming counterparts.

Previous work. Feigenbaum et al. [4] were the first to study matching problems under similar assumptions. Their main results in this context were a semi-streaming algorithm that computes a $(3/2 + \varepsilon)$-approximation in $O(\log(1/\varepsilon)/\varepsilon)$ passes for maximum cardinality matching in bipartite graphs, as well as a one-pass 6-approximation for maximum weighted matching in arbitrary graphs. Later on, McGregor [12] improved on these findings, to obtain performance guarantees of $(1 + \varepsilon)$ and $(2 + \varepsilon)$ for the maximum cardinality and maximum weight versions, respectively, being able to handle arbitrary graphs with only a constant number of passes (depending on $1/\varepsilon$). In addition, McGregor [12] tweaked the one-pass algorithm of Feigenbaum et al. into achieving a ratio of 5.828. Finally, Zelke [17] has recently attained an improved approximation factor of 5.585, which stands as the currently best one-pass algorithm. Note that the 6-approximation algorithm in [4] and the 5.828-approximation algorithm in [12] are preemptive online algorithms. On the other hand, the algorithm of Zelke [17] uses the notion of shadow-edges, which may be re-inserted into the matching, and hence it is not an online algorithm.

Main result I. The first contribution of this paper is to improve on the above-mentioned results, by devising a deterministic one-pass algorithm in the semi-streaming model, whose performance guarantee is $4.91 + \varepsilon$. In a nutshell, our approach is based on partitioning the edge set into $O(\log n)$ weight classes, and computing a separate maximal matching for each such class in online fashion, using $O(n \cdot \text{polylog}(n))$ memory bits overall. The crux lies in proving that the union of these matchings contains a single matching whose weight compares favorably to the optimal one. The specifics of this algorithm are presented in Section 2.

Main result II. Our second contribution is motivated by the relation between semi-streaming algorithms and *preemptive* online algorithms, which must maintain a feasible matching at any point in time. To our knowledge, there are currently no lower bounds on the competitive ratio that can be achieved by incorporating preemption. Thus, we also provide a lower bound of 4.967 on the performance guarantee of any such deterministic algorithm. As a result, we show that improved one pass algorithms for this problem must store more than just a matching in memory. Further details are provided in Section 3.

Main result III. We conclude with the first ever experimental study in the context of semi-streaming algorithms for matching problems, conducted in order to compare the practical performance of our approach to that of previously suggested algorithms. In Section 4, we demonstrate that by carefully calibrating some cut-off parameters, combined with the idea of running multiple algorithms in parallel, one can achieve practical performance guarantees that far exceed theoretical ones, at least when real-life instances are considered.

2. The Semi-Streaming Algorithm

This section is devoted to obtaining main result I, that is, an improved one-pass algorithm for the weighted matching problem in the semi-streaming model. We begin by presenting a simple deterministic algorithm with a performance guarantee of 8. We then show how to randomize its parameters, still within the semi-streaming framework, and obtain an expected approximation ratio of 4.9108. Finally, we de-randomize the algorithm by showing how to emulate the required randomness using multiple copies (constant number) of the deterministic algorithm, while paying an additional additive factor of at most ε, for any fixed $\varepsilon > 0$.

2.1. A simple deterministic approach

Preliminaries. We maintain the maximum weight of any edge $w_{\max}$ seen so far in the input stream. Clearly, the maximum weight matching of the edges seen so far has weight in the interval $\left[w_{\max}, \frac{n}{2} w_{\max}\right]$. We denote a maximum weight matching and its cost by $\textsc{opt}$; it should be clear which one is meant from the context. Note that if we disregard all edges with weight at most $\frac{2\tilde{\varepsilon} w_{\max}}{n}$, the weight of the maximum weight matching in the resulting instance decreases by an additive term of at most $\tilde{\varepsilon}\, w_{\max} \le \tilde{\varepsilon}\, \textsc{opt}$.

Our algorithm has a parameter $\gamma > 1$, and a value $\phi > 0$. We define weight classes of edges in the following way. For every $i \in \mathbb{Z}$, we let the class W_i be the collection of edges whose weight is in the interval $\left[\phi\gamma^i, \phi\gamma^{i+1}\right)$. We note that by our initial assumption, the weight of each edge is in the interval $\left[\frac{2\tilde{\varepsilon} w_{\max}}{n}, w_{\max}\right]$, and we say that a weight class W_i is *under consideration* if its weight interval $\left[\phi\gamma^i, \phi\gamma^{i+1}\right)$ intersects $\left[\frac{2\tilde{\varepsilon} w_{\max}}{n}, w_{\max}\right]$. The number of classes which are under consideration at any point in time is $O\left(\log_\gamma \frac{n}{\tilde{\varepsilon}}\right)$.

The algorithm. Our algorithm simply maintains the list of classes under consideration and maintains a maximal (unweighted) matching for each such class. In other words, when the value of $w_{\max}$ changes, we delete from the memory some of these matchings, corresponding to classes that are no longer under consideration. Note that to maintain a maximal matching in a given subgraph, we only need to check if the two endpoints of the new edge are not covered by existing edges of the matching.

To conclude, for every new edge $e \in E$ we proceed as follows. We first check if $w(e)$ is greater than the current value of $w_{\max}$. If so, we update $w_{\max}$ and the list of weight classes under consideration accordingly. Then, we find the weight class of $w(e)$, and try to extend its corresponding matching; i.e., e will be added to this matching if it remains a matching after doing so.

Note that at each point the content of the memory is comprised of a fixed number of parameter values and a collection of $O\left(\log_\gamma \frac{n}{\tilde{\varepsilon}}\right)$ matchings, consisting of $O\left(n \log_\gamma \frac{n}{\tilde{\varepsilon}}\right)$ edges overall. Therefore, our algorithm indeed falls in the semi-streaming model.

At the conclusion of the input sequence, we need to return a single matching rather than a collection of matchings. To this end, we could compute a maximum weighted matching of the edges in the current memory. However, for the specific purposes of our analysis, we use the following faster algorithm. We sort the edges in memory in decreasing order of weight classes, such that the edges in W_i appear before those in W_{i-1}, for every i. Using this sorted list of edges, we apply a greedy algorithm for selecting a maximal matching, in which the current edge is added to this matching if it remains a matching after doing so. Then, the post-processing time needed is linear in the size of the memory used, that is, $O\left(n \log_\gamma \frac{n}{\tilde{\varepsilon}}\right)$. This concludes the presentation of the algorithm and its implementation as a semi-streaming algorithm.

Analysis. For purposes of analysis, we round down the weight of each edge so that the weight of all edges in W_i equals $\phi\gamma^i$. This way, we obtain *rounded* edge weights. For our optimal solution $\textsc{opt}$ let us denote by $\textsc{opt}'$ its rounded weight. The next claim follows from the definition of W_i.

Lemma 2.1. $\textsc{opt} \leq \gamma\, \textsc{opt}'$.

As an intermediate step, we analyze an improved algorithm that keeps all weight classes. That is, for each i, we use M_i to denote the maximal matching of class W_i at the end of the input, and denote by M the solution obtained by this algorithm, if we would have applied it. Similarly, we denote by $\textsc{opt}_i$ the set of edges in $\textsc{opt}$ which belong to W_i. For every i, we define the set of vertices P_i, associated with W_i, to be the set of endpoints of edges in M_i that are not associated with higher weight classes:

$$P_i = \{\, u, v \mid (u, v) \in M_i\} \setminus (P_{i+1} \cup P_{i+2} \cup \cdots).$$

For a vertex $p \in P_i$, we define its associated weight to be $\phi\gamma^i$. For vertices which do not belong to any P_i, we let their associated weight be zero. We next bound the total associated weight of all vertices.

Lemma 2.2. *The total associated weight of all vertices is at most* $\frac{2\gamma}{\gamma-1} \cdot w(M)$.

Proof. Consider a vertex $u \in P_i$ and let (u, v) be the edge in M_i adjacent to u. If $(u, v) \in M$ then we charge the weight associated with u to the edge (u, v). Thus, an edge $e \in M_i$ is charged at most twice from vertices associated with its own weight class. Otherwise, if $(u, v) \notin M$ then there must be some other edge $e \in M \cap M_j$, for some $j > i$, that prevented us from adding (u, v) to M, in which case we charge the weight associated with u to e. Notice that $u \notin e$, or otherwise, u would not be associated with W_i. Thus, the edge $e \in M_j$ must be of the form $e = (v, x)$ and can only be charged twice from vertices in weight class i, once through v and once through x.

To bound the ratio between $w(M)$ and the total associated weight of the vertices, it suffices to bound the ratio between the weight of an edge $e \in M$ and the total associated weight of the vertices which are charged to e. Assume that $e \in M_j$, then there are at most two vertices which are charged to e and class i for all $i \leq j$, and no vertex is associated to e and class i for $i > j$. Hence, the total associated weight of these vertices is at most

$$2 \sum_{i \leq j} \phi\gamma^i < 2\phi\gamma^j \cdot \sum_{i=0}^{\infty} \frac{1}{\gamma^i} = 2\phi\gamma^j \cdot \frac{1}{1 - 1/\gamma} = \phi\gamma^j \cdot \frac{2\gamma}{\gamma - 1},$$

and the claim follows since $w(e) \geq \phi\gamma^j$. ■

It remains to bound $\textsc{opt}'$ with respect to the total associated weight.

Lemma 2.3. *The total weight associated with all vertices is at most* $\textsc{opt}'$.

Proof. It suffices to show that for every edge $e = (x, y) \in \textsc{opt}_i$ the maximum of the associated weights of x and y is at least the rounded weight of e. Suppose that this claim does not hold, then x and y are not covered by M_i, as otherwise their associated weight would be at least $\phi\gamma^i$. Hence, when the algorithm considered e, we would have added e to M_i, contradicting our assumption that x and y are not covered by M_i. ■

Now instead of considering all weight categories, we construct the matching M only using edges with weight at least $\frac{2\tilde{\varepsilon} w_{\max}}{n}$. Using the above sequence of lemmas, and recalling that we lose another $\frac{1}{1-\tilde{\varepsilon}}$ factor in the approximation ratio due to disregarding these cheap edges, we obtain the following inequality:

$$\textsc{opt} \leq \gamma\textsc{opt}' \leq \frac{1}{1 - \tilde{\varepsilon}} \cdot \frac{2\gamma^2}{\gamma - 1} \cdot w(M). \tag{2.1}$$

For any $\varepsilon > 0$, setting $\tilde{\varepsilon} = \frac{\varepsilon}{\left(2\gamma^2/(\gamma-1)\right)+\varepsilon}$, we get an approximation ratio of $\frac{2\gamma^2}{\gamma-1} + \varepsilon$. This ratio is optimized for $\gamma = 2$, where it equals $(8 + \varepsilon)$. Hence, we have established the following theorem.

Theorem 2.4. *For any fixed* $\varepsilon > 0$, *there is a deterministic one-pass semi-streaming algorithm whose approximation ratio is* $8 + \varepsilon$.

2.2. Improved approximation ratio through randomization

In what follows, we analyze a randomized variant of the deterministic algorithm which was presented in the previous subsection. In general, this variant sets the value of ϕ to be $\phi = \gamma^\delta$ where δ is a random variable. This method is commonly referred to as *randomized geometric grouping* [8].

Formally, let δ be a continuous random variable which is uniformly distributed on the interval $[0, 1)$. We define the weight class $W_i(\delta)$ to be the edges whose weight is in the interval $\left[\gamma^{i+\delta}, \gamma^{i+1+\delta}\right)$, and run the algorithm as in the previous subsection. Note that this algorithm uses only the partition of the edges into classes and not the precise values of their weights. In addition, we denote by $M(\delta)$ the resulting matching obtained by the algorithm, and by $TW(\delta)$ the total associated weight of the vertices, where for a vertex $p \in P_i$ we define its associated weight to be $\gamma^{i+\delta}$; i.e., the minimal value in the interval defining $W_i(\delta)$. We also denote by $\textsc{opt}'(\delta)$ the value of $\textsc{opt}'$ for this particular δ.

For any fixed value of δ, inequality (2.1) immediately implies $\textsc{opt}'(\delta) \leq \left(\frac{2\gamma}{\gamma-1} + \varepsilon\right) \cdot w(M(\delta))$. Note that $\textsc{opt}'(\delta)$ and $w(M(\delta))$ are random variables, such that for each realization of δ the above inequality holds. Hence, this inequality holds also for their expected values. That is, we have established the following lemma where $\mathrm{E}_\delta[\cdot]$ represents expectation with respect to the random variable δ.

Lemma 2.5. $\mathrm{E}_\delta[\textsc{opt}'(\delta)] \le \left(\frac{2\gamma}{\gamma-1} + \varepsilon\right) \cdot \mathrm{E}_\delta[w(M(\delta))]$.

We next upper bound $\textsc{opt}$ in terms of $\mathrm{E}_\delta[\textsc{opt}'(\delta)]$.

Lemma 2.6. $\frac{\gamma \ln \gamma}{\gamma-1} \cdot \mathrm{E}_\delta[\textsc{opt}'(\delta)] \ge \textsc{opt}$.

Proof. We will show the corresponding inequality for each edge $e \in \textsc{opt}$. We denote by $w'_\delta(e)$ the rounded weight of e for a specific value of δ. Then, it suffices to show that $\frac{\gamma \ln \gamma}{\gamma-1} \cdot \mathrm{E}_\delta[w'_\delta(e)] \ge w(e)$. Let p be an integer, and let $0 \le \alpha < 1$ be the value that satisfies $w(e) = \gamma^{p+\alpha}$. Then, for $\delta \le \alpha$, $w'_\delta(e) = \gamma^{p+\delta}$, and for $\delta > \alpha$, $w'_\delta(e) = \gamma^{p-1+\delta}$, thus the expected rounded weight of e over the choices of δ is

$$\mathrm{E}_\delta[w'_\delta(e)] = \int_0^\alpha \gamma^{p+\delta} d\delta + \int_\alpha^1 \gamma^{p-1+\delta} d\delta = \frac{1}{\ln \gamma} \cdot \left(\gamma^p(\gamma^\alpha - 1) + \gamma^{p-1}(\gamma - \gamma^\alpha)\right) = w(e) \cdot \left(1 - \frac{1}{\gamma}\right) \frac{1}{\ln \gamma},$$

and the claim follows. ■

Combining the above two lemmas we obtain that the expected weight of the resulting solution is at least $\left(\frac{(\gamma-1)^2}{2\gamma^2 \ln \gamma} + \varepsilon\right) \cdot \textsc{opt}$. This approximation ratio is optimized for $\gamma \approx 3.513$, where it is roughly $(4.9108 + \varepsilon)$. Hence, we have established the following theorem.

Theorem 2.7. *For any fixed $\varepsilon > 0$, there is a randomized one-pass semi-streaming algorithm whose expected approximation ratio is roughly* $4.9108 + \varepsilon$.

2.3. Derandomization

Prior to presenting our de-randomization, we slightly modify the randomized algorithm of the previous subsection. In this variation, instead of picking δ uniformly at random from the interval $[0, 1)$ we pick δ' uniformly at random from the discrete set $\left\{0, \frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\right\}$, where q is a parameter whose value will be determined later. We apply the same method as in the previous section, replacing δ by δ'. Then, using Lemma 2.5, we obtain $\mathrm{E}_{\delta'}[\textsc{opt}'(\delta')] \le \left(\frac{2\gamma}{\gamma-1} + \varepsilon\right) \cdot \mathrm{E}_{\delta'}[w(M(\delta'))]$. To extend Lemma 2.6 to this new setting, we note that δ' can be obtained by first picking δ and then rounding it down to the largest number in $\left\{0, \frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\right\}$ which is at most δ. In this way, we couple the distributions of δ and δ'. Now consider the rounded weight of an edge e in $\textsc{opt}$ in the two distinct values of δ and δ'. The ratio between the two rounded weights is at most $\gamma^{1/q}$. Therefore, we establish that $\frac{\gamma \ln \gamma}{\gamma-1} \cdot \gamma^{1/q} \cdot \mathrm{E}_{\delta'}[\textsc{opt}'(\delta')] \ge \textsc{opt}$, and the resulting approximation ratio of the new variation is $\frac{2\gamma^{2+1/q} \ln \gamma}{(\gamma-1)^2} + \varepsilon$. By setting q to be large enough (picking $q = \lceil \log_\gamma^{-1}(\varepsilon/5) \rceil$ is sufficient), the resulting approximation ratio is bounded by $\frac{2\gamma^2 \ln \gamma}{(\gamma-1)^2} + 2\varepsilon$.

De-randomizing the new variation in the semi-streaming model is straightforward. We simply run in parallel all q possible outcomes of the algorithm, one for each possible value of δ', and pick the best solution among the q solutions obtained. Since q is a constant (for fixed values of ε), the resulting algorithm is still a semi-streaming algorithm whose performance guarantee is $4.9108 + 2\varepsilon$. By scaling ε prior to applying the algorithm, we establish the following result.

Theorem 2.8. *For any fixed $\varepsilon > 0$, there is a deterministic one-pass semi-streaming $(4.9108 + \varepsilon)$-approximation algorithm for the weighted matching problem. This algorithm processes each input edge in constant time and required $O(n)$ time at the end of the input to compute the final output.*

3. Online Preemptive Matching

In this section, we establish the following theorem.

Theorem 3.1. *The competitive ratio of any deterministic preemptive online algorithm is at least $\mathcal{R} - \varepsilon$ for any $\varepsilon > 0$, where $\mathcal{R} \approx 4.967$ is the unique real solution of the equation $x^3 = 4(x^2 + x + 1)$.*

Recall that the algorithm of Feigenbaum et al. [4] and that of McGregor [12] can be viewed as online preemptive algorithms; their competitive ratios are 6 and 5.828, respectively.

Definitions and properties. Let $C = \mathcal{R} - \varepsilon$ for some arbitrary but fixed $\varepsilon > 0$. Our goal is to show that the competitive ratio of any deterministic algorithm is at least C. To this end, we construct an input graph iteratively. In the construction of the input, edge weights come from two sequences. The main sequence $w_1, w_2, \ldots$, and the additional sequence is $w'_1, w'_2, \ldots$, are defined as follows:

$$w_i = \begin{cases} 1 & i=1 \\ \frac{1}{2C+1}\left((C^2+1)w_{i-1} - C\sum_{j=1}^{i-2} w_j\right) & i>1 \end{cases} \qquad w'_i = \begin{cases} 1 & i=1 \\ \frac{1}{C}\left((C+1)w_i - w_{i-1}\right) & i>1 \end{cases} \tag{3.1}$$

The sequences are defined according to (3.1) as long as $w_{i-1} \geq w_{i-2}$. As soon as $w_{n-1} < w_{n-2}$ for some n, both sequences stop with w_n and w'_n, respectively. In the full version [3] of this paper we show that the sequences are well defined in the sense that they indeed have finite length. Let $S_i = \sum_{j=1}^{i} w_j$ and $S_0 = 0$.

From the definition (3.1) and simple algebra, one can derive the following properties of these sequences. We omit their justification due to lack of space.

Property 1. For all $i = 1, \ldots, n-2$ we have $w_i \leq w'_i$, but $w_{n-1} > w'_{n-1}$.

Property 2. For all $i = 1, \ldots, n-2$ we have $Cw_i = S_{i-1} + w_{i+1} + w'_{i+1}$.

Property 3. For all $i = 1, \ldots, n-2$ we have $Cw'_i = S_{i-2} + w_i + w_{i+1} + w'_{i+1}$.

Input construction, step 1. To better understand our construction, we advise the reader to consult Figure 1. The input is created in n steps. In the initial step, two edges (a_1, x_1) and (b_1, x_1), each of weight w_1, are introduced. Assume that after both edges have arrived, the online algorithm keeps the edge (a_1, x_1).

Every future step can be of two distinct types, which will be described later on. We maintain the following invariants throughout the construction.

Invariant 1. Immediately after the ith step, the set $M_i = \{(x_1, b_1), \ldots, (x_i, b_i)\}$ forms a matching.

Invariant 2. Immediately after the ith step, the algorithm keeps a single edge e_i, which can be one of two edges:

i) If $e_i = (x_i, a_i)$ then its weight is w_i and a_i is unmatched in M_i.
ii) If $e_i = (y_i, c_i)$ then its weight is w'_i, $y_i = x_j$ for some $j < i$, and c_i is unmatched in M_i.

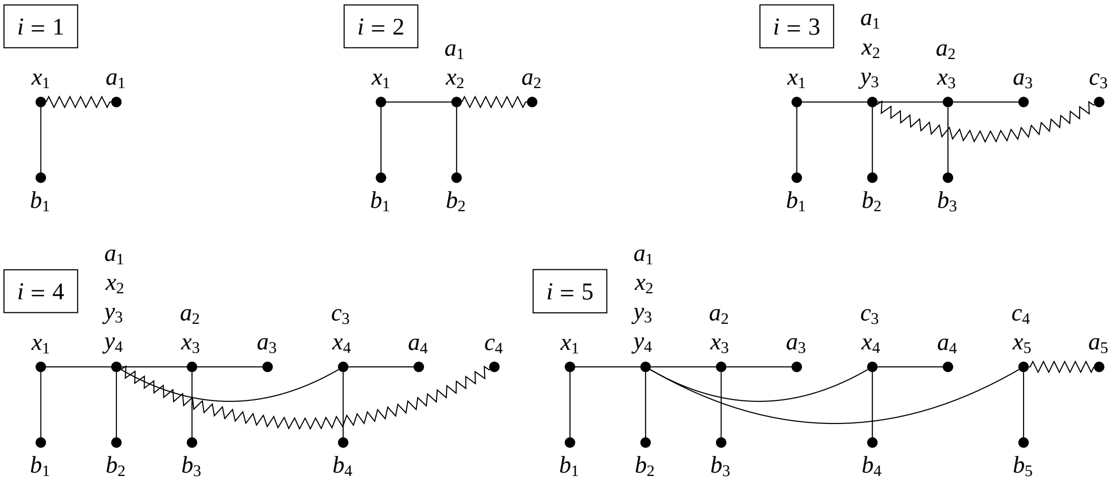


Figure 1: An example of five steps of the lower bound construction. The curved edges denote the edge kept by the online algorithm at each time. In the first two steps, the edges (x_i, a_i) are chosen by the algorithm. In the third step, (x_3, a_3) is not chosen by the algorithm, so (y_3, c_3) arrives next. In the fourth step, (x_4, a_4) is not chosen by the algorithm, so (y_4, c_4) arrives next. In the fifth step, (x_5, a_5) is chosen by the algorithm, so no further edges arrive in this step.

The invariants clearly holds after the first step: The algorithm keeps (x_1, a_1) and a_1 is free in $M_1 = \{(x_1, b_1)\}$. We next define the subsequent steps and show that the invariants holds throughout.

Input construction, step $i + 1 < n$. We now show how to construct the edges of step $i + 1$, for the case $i + 1 < n$. We introduce two new edges of weight w_{i+1}. Let x_{i+1} be a_i if $e_i = (x_i, a_i)$, and x_{i+1} be c_i if $e_i = (y_i, c_i)$. The new edges are (x_{i+1}, b_{i+1}), and (x_{i+1}, a_{i+1}), where a_{i+1} and b_{i+1} are new vertices. According to Invariant 2, the vertex x_{i+1} is unmatched in M_i. It follows that M_{i+1} is a matching and thus Invariant 1 holds in this step. Both edges have a common endpoint with the edge that the algorithm has, and the algorithm can either preempt e_i, in which case we assume (without loss of generality) that it now has (x_{i+1}, a_{i+1}), or else it keeps the previous edge. If the algorithm holds onto e_i then let y_{i+1} be x_i if $e_i = (x_i, a_i)$, and y_{i+1} be y_i if $e_i = (y_i, c_i)$. In this case a third edge, (y_{i+1}, c_{i+1}), with weight of w'_{i+1} is introduced. The vertex c_{i+1} is new. There are four cases to consider depending on which edge the algorithm had at the end of the ith step and whether it is preempted right away or not.

In the first case, the algorithm has $e_i = (x_i, a_i)$ at the end of the ith step and replaces it with $(x_{i+1}, a_{i+1}) = (a_i, a_{i+1})$. Since a_{i+1} is a new vertex (and different than x_{i+1}) it follows that a_{i+1} is free in M_{i+1}. Thus, case i) of Invariant 2 holds.

In the second case, the algorithm has $e_i = (y_i, c_i)$ at the end of the ith step and it replaces it with $(x_{i+1}, a_{i+1}) = (c_i, a_{i+1})$. It follows that a_{i+1} is free in M_{i+1}. Thus, case ii) of Invariant 2 holds.

For the remaining two cases note that if $w'_i \leq 0$ or $w_i < 0$ and the algorithm has a single edge of weight w'_i or w_i, respectively, then the optimal solution is strictly positive and the value of the algorithm is non-positive, hence the resulting approximation ratio in this case is unbounded.

Consequently, we can assume without loss of generality that if the algorithm has a single edge at the end of step i then its weight is strictly positive.

Now consider the case where the algorithm has $e_i = (a_i, x_i)$ at the end of the ith step but does not replace it with $(x_{i+1}, a_{i+1}) = (a_i, a_{i+1})$. If this happens, we show that the algorithm must replace the edge with $(y_{i+1}, c_{i+1}) = (x_i, c_{i+1})$. Assume that this is not the case. Then the profit of the algorithm is w_i. Consider the solution $M_{i+1} - (x_i, b_i) + (y_{i+1}, c_{i+1})$. The cost of this matching is $S_{i+1} - w_i + w'_{i+1}$, which equals Cw_i, by Property 2. In other words, the solution kept by the algorithm is C-competitive. Since our goal is to prove precisely this, we can assume this event never happens. Thus, the algorithm must switch to the edge (y_{i+1}, c_{i+1}), which leads us to case ii) of Invariant 2.

Finally, consider the case where the algorithm has $e_i = (y_i, c_i)$ at the end of the ith step but does not replace it with $(x_{i+1}, a_{i+1}) = (c_i, a_{i+1})$. If this happens, we show that the algorithm must replace the edge with $(y_{i+1}, c_{i+1}) = (y_i, c_{i+1})$. Assume that this is not the case. Then the profit of the algorithm is w'_i. Consider the solution $M_{i+1} - (x_j, b_j) + (y_{i+1}, c_{i+1})$, where $j < i$ is the index from case ii) in Invariant 2 that corresponds to e_i. The cost of this matching is at least $S_{i+1} - w_{i-1} + w'_{i+1}$, which equals Cw'_i, by Property 3. As in the previous case, we can assume this never happens. Thus, the algorithm must switch to the edge (x_{i+1}, a_{i+1}), which leads us to case ii) of Invariant 2.

This finishes the description of the input graph construction, as well as the justification that Invariants 1 and 2 hold at each step along the way.

Bounding the competitive ratio. We next define a recursive formula for S_i. By definition (3.1) of the sequence w_i, we have

$$\begin{cases} S_0 = 0 \\ S_1 = 1 \\ S_{k+1} = \frac{C^2+2C+2}{2C+1} S_k - \frac{C^2+C+1}{2C+1} S_{k-1}, \quad \text{for } k \geq 1 \end{cases} \tag{3.2}$$

Lemma 3.2. *There exists a value of n such that $w_{n-2} > w_{n-1}$; for this value, $\frac{S_{n-1}}{w_{n-1}} > C$ holds.*

Proof. The first claim is proved by solving the recurrence (3.2), using standard tools [3]. To prove the second part, note that $w_{n-2} > w_{n-1}$ is equivalent to $S_{n-1} - S_{n-2} < S_{n-2} - S_{n-3}$. Hence using the recursive formula we conclude that

$$S_{n-1} - 2S_{n-2} + \frac{2C + 1}{-C^2 - C - 1} S_{n-1} + \frac{C^2 + 2C + 2}{C^2 + C + 1} S_{n-2} < 0,$$

that is,

$$S_{n-1} \cdot (C^2 + C + 1 - 2C - 1) + S_{n-2} \cdot (C^2 + 2C + 2 - 2C^2 - 2C - 2) < 0,$$

which is equivalent to $(C^2 - C)S_{n-1} - C^2 S_{n-2} < 0$, so $C(S_{n-1} - S_{n-2}) < S_{n-1}$, and we conclude that $Cw_{n-1} < S_{n-1}$, as claimed. ■

Everything is in place to prove the main claim of this section.

Proof of Theorem 3.1. From Invariant 2, we conclude that at the end of iteration $n-1$, the algorithm only has the edge e_{n-1}, which can have weight w'_{n-1} or w_{n-1}. From Property 1, it follows that $\max\{w_{n-1}, w'_{n-1}\} = w_{n-1}$. On the other hand, from Invariant 1, we know that there is a matching with cost S_{n-1}. Therefore, the competitive ratio of any algorithm is at least $\frac{S_{n-1}}{w_{n-1}}$. From Lemma 3.2 we then conclude that the competitive ratio is at least C. ■

4. Experimental Evaluation

In this section, we present the results of an empirical study, conducted in order to compare the practical performance of our approach to that of previously suggested algorithms. More specifically, the complete set of algorithms that have been implemented and extensively tested can be briefly listed as follows:

- LAYERED: The algorithm described in Section 2, which keeps $O\left(\log \frac{n}{\epsilon}\right)$ matchings.
- ONLINE: The algorithm of McGregor [12], based on keeping a single matching at all times.
- SHADOW: The algorithm of Zelke [17], with two shadow edges for each matching edge.

For LAYERED and SHADOW, we made use of an addition optimization phase, in which a maximum weight matching is computed among the edges that were kept in memory. The main reason for this extra effort is that we were interested in determining the best possible practical performance that can be extracted out of these algorithms, rather than in worse case performance and nothing more. We point out that this phase is performed only once, and that one can always employ a linear-time approximation [11] should running time be a concern.

Special features. Each of above-mentioned algorithms is parameterized. Typically, this parameter is chosen to minimize the worst-case approximation ratio obtained in theory. However, this choice need not be the one leading to the best performance in practice. Therefore, we considered three versions of each algorithm, with different parameters: one was chosen empirically to obtain best possible guarantees; another is the value emanating from the theoretical analysis; and the last one is just averaging these two. We also examined the consequences of combining different algorithms. Under this scheme, all algorithms are executed in parallel and, at the end, a maximum weight matching is computed with respect to the collection of edges kept by any of these algorithms.

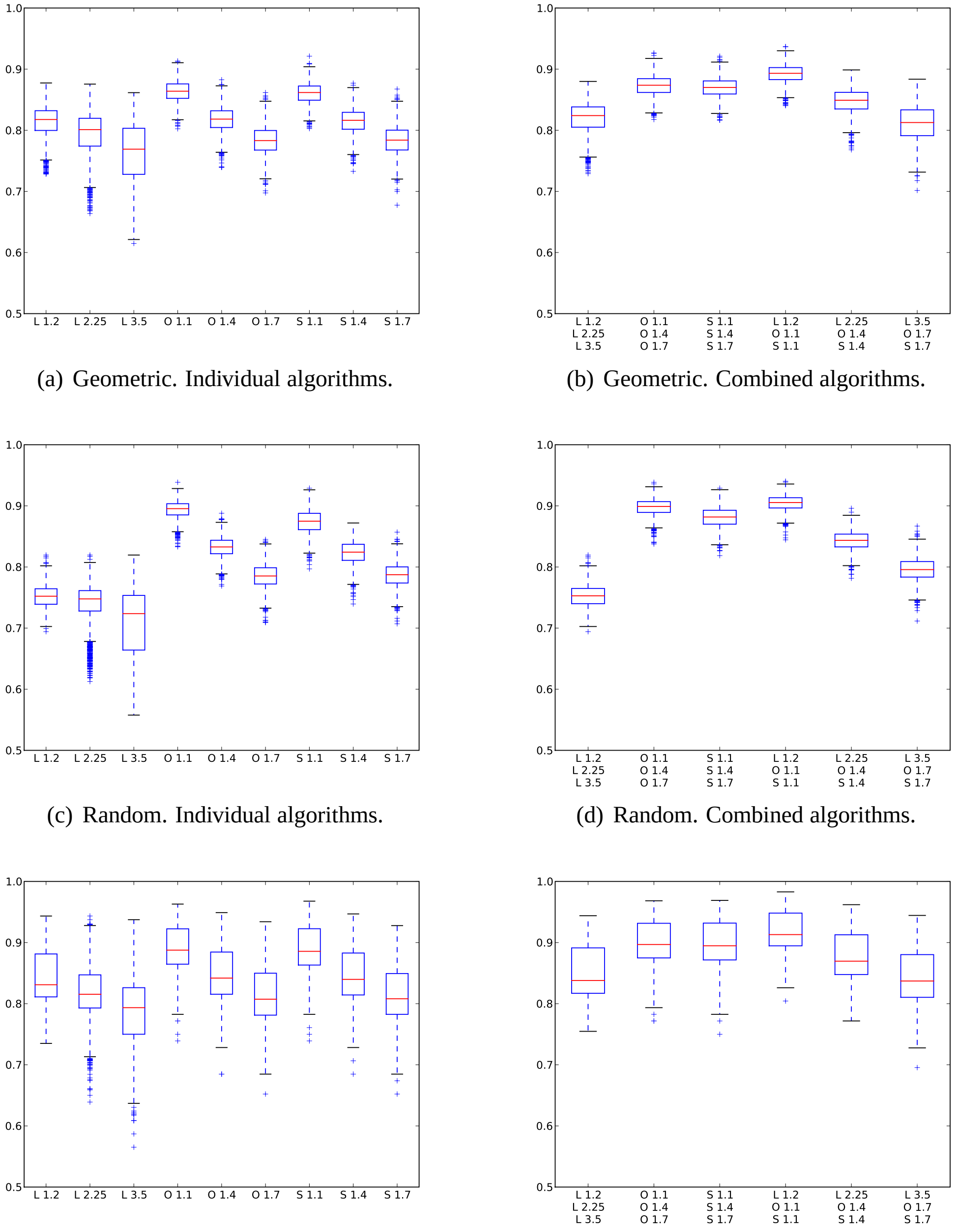
Actual tests performed. We evaluated LAYERED, ONLINE, and SHADOW with test graphs of roughly 1000 vertices. Following the approach of previous experimental papers in this context [13, 11], we investigated three different classes of graphs:

- Geometric: Points were drawn uniformly at random from the unit square; the weight of an edge is the Euclidean distance between its endpoints.
- Real world: Points are taken from geometric instances in the TSPLIB [15]; once again, edge weights are determined by Euclidean distances.
- Random: The weight of each edge is an integer picked uniformly and independently at random from $1, \ldots, |V|$.

From each of these classes, we generated 10 base instances. In addition, as the performance of all algorithms under consideration depends heavily on the particular order by which edges are revealed, each algorithm was tested on every base instance for 200 independent runs, with a random edge permutation each time. To speed up the experiments all graphs were sparsified by keeping, for each vertex, the connections to one third of its closest nodes. The results are presented in Figure 2.

Conclusions. One can notice right away that the algorithms in question perform significantly better when their respective parameters are set considerably lower than the best theoretical value (1.2 for LAYERED, and 1.1 for ONLINE and SHADOW). With this optimization in place, it appears that ONLINE and SHADOW have comparable performance, but outperform LAYERED.

Regarding the combination of several algorithms, we compared for each algorithm the combined output of its three versions (depending on parameter setting) and the outcome of combining the best version of each of the three algorithms. We consistently observed that it is preferable to combine the output of completely different algorithms rather than the same algorithm with different parameters.



(a) Geometric. Individual algorithms.

(b) Geometric. Combined algorithms.

(c) Random. Individual algorithms.

(d) Random. Combined algorithms.

(e) Real. Individual algorithms.

(f) Real. Combined algorithms.

Figure 2: The results of individual algorithms appear on the left column, while the performance of combining them is shown on the right. The algorithms are specified as x-axis labels using first letters (L for LAYERED, O for ONLINE, and S for SHADOW), followed by the precise parameter value for that version. Box plots describing the outcome of our experiments are given above. Each box contains outcomes with performance between the .25 and .75 quartile, where the horizontal line inside designated the median.

Finally, we point out that, as it is often the case for approximation algorithms, the observed performance of all algorithms is significantly better than the theoretical worst case guarantee. It is worth noting, however, that their performance is still worse than traditional heuristics (such as the greedy algorithm) that are not constrained by the extent of memory usage. For example, in geometric graphs, these heuristics can recover on average 99% of the optimal value [11], whereas none of the individual algorithms can recover more than 90%.

References

[1] N. Bansal, N. Buchbinder, A. Gupta, and J. Naor. An $O(\log^2 k)$-competitive algorithm for metric bipartite matching. In *Proceedings of the 15th Annual European Symposium on Algorithms*, pages 522–533, 2007.

[2] M. Elkin and J. Zhang. Efficient algorithms for constructing $(1 + \epsilon, \beta)$-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006.

[3] L. Epstein, A. Levin, J. Mestre, and D. Segev. Improved approximation guarantees for weighted matching in the semi-streaming model, 2009. http://arxiv.org/abs/0907.0305.

[4] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

[5] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2008.

[6] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. *Journal of Algorithms*, 23(1):180–194, 1997.

[7] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

[8] M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–79, 1996.

[9] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 352–358, 1990.

[10] S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.

[11] J. Maue and P. Sanders. Engineering algorithms for approximate weighted matching. In *Proceedings of the 6th International Workshop on Experimental Algorithms*, pages 242–255, 2007.

[12] A. McGregor. Finding graph matchings in data streams. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 170–181, 2005.

[13] M. Müller-Hannemann and A. Schwartz. Implementing weighted b-matching algorithms: Towards a flexible software design. *ACM Journal on Experimental Algorithmics*, 4:7, 1999.

[14] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Foundations and Trends in Theoretical Computer Science. Now Publishers Inc, 2005.

[15] G. Reinelt. TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[16] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[17] M. Zelke. Weighted matching in the semi-streaming model. In *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science*, pages 669–680, 2008.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 359-370
www.stacs-conf.org

COMPUTING LEAST FIXED POINTS OF PROBABILISTIC SYSTEMS OF POLYNOMIALS

JAVIER ESPARZA AND ANDREAS GAISER AND STEFAN KIEFER

Fakultät für Informatik, Technische Universität München, Germany
E-mail address: {esparza,gaiser,kiefer}@model.in.tum.de

ABSTRACT. We study systems of equations of the form $X_1 = f_1(X_1, \ldots, X_n), \ldots, X_n = f_n(X_1, \ldots, X_n)$ where each f_i is a polynomial with nonnegative coefficients that add up to 1. The least nonnegative solution, say μ, of such equation systems is central to problems from various areas, like physics, biology, computational linguistics and probabilistic program verification. We give a simple and strongly polynomial algorithm to decide whether $\mu = (1, \ldots, 1)$ holds. Furthermore, we present an algorithm that computes reliable sequences of lower and upper bounds on μ, converging linearly to μ. Our algorithm has these features despite using inexact arithmetic for efficiency. We report on experiments that show the performance of our algorithms.

1. Introduction

We study how to efficiently compute the least nonnegative solution of an equation system of the form

$$X_1 = f_1(X_1, \ldots, X_n) \quad \ldots \quad X_n = f_n(X_1, \ldots, X_n)\ ,$$

where, for every $i \in \{1, \ldots, n\}$, f_i is a polynomial over $X_1, \ldots, X_n$ with positive rational coefficients that *add up to 1*.[1] The solutions are the fixed points of the function $f : \mathbb{R}^n \to \mathbb{R}^n$ with $f = (f_1, \ldots, f_n)$. We call f a *probabilistic system of polynomials* (short: *PSP*). E.g., the PSP

$$f(X_1, X_2) = \left(\frac{1}{2}X_1X_2 + \frac{1}{2}\ ,\ \frac{1}{4}X_2X_2 + \frac{1}{4}X_1 + \frac{1}{2} \right)$$

induces the equation system

$$X_1 = \tfrac{1}{2}X_1X_2 + \tfrac{1}{2} \qquad X_2 = \tfrac{1}{4}X_2X_2 + \tfrac{1}{4}X_1 + \tfrac{1}{2}\ .$$

Obviously, $\overline{1} = (1, \ldots, 1)$ is a fixed point of every PSP. By Kleene's theorem, every PSP has a least nonnegative fixed point (called just least fixed point in what follows), given by the limit of the sequence $\overline{0}, f(\overline{0}), f(f(\overline{0})), \ldots$

PSPs are important in different areas of the theory of stochastic processes and computational models. A fundamental result of the theory of branching processes, with numerous applications in physics, chemistry and biology (see e.g. [9, 2]), states that extinction probabilities of species are

1998 ACM Subject Classification: F.2.1 Numerical Algorithms and Problems, G.3 Probability and Statistics.
Key words and phrases: computing fixed points, numerical approximation, stochastic models, branching processes.

[1] Later, we allow that the coefficients add up to *at most* 1.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2468

equal to the least fixed point of a PSP. The same result has been recently shown for the probability of termination of certain probabilistic recursive programs [7, 6]. The consistency of stochastic context-free grammars, a problem of interest in statistical natural language processing, also reduces to checking whether the least fixed point of a PSP equals $\overline{1}$ (see e.g. [11]).

Given a PSP f with least fixed point μ_f, we study how to efficiently solve the following two problems: (1) decide whether $\mu_f = \overline{1}$, and (2) given a rational number $\epsilon > 0$, compute $\mathbf{lb}, \mathbf{ub} \in \mathbb{Q}^n$ such that $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$ (where $\mathbf{u} \leq \mathbf{v}$ for vectors $\mathbf{u}, \mathbf{v}$ means $\leq$ in all components). While the motivation for Problem (2) is clear (compute the probability of extinction with a given accuracy), the motivation for Problem (1) requires perhaps some explanation. In the case study of Section 4.3 we consider a family of PSPs, taken from [9], modelling the neutron branching process in a ball of radioactive material of radius D (the family is parameterized by D). The least fixed point is the probability that a neutron produced through spontaneous fission *does not* generate an infinite "progeny" through successive collisions with atoms of the ball; loosely speaking, this is the probability that the neutron *does not* generate a chain reaction and the ball *does not* explode. Since the number of atoms in the ball is very large, spontaneous fission produces many neutrons per second, and so even if the probability that a given neutron produces a chain reaction is very small, the ball will explode with large probability in a very short time. It is therefore important to determine the largest radius D at which the probability of no chain reaction is still 1 (usually called the *critical radius*). An algorithm for Problem (1) allows to compute the critical radius using binary search. A similar situation appears in the analysis of parameterized probabilistic programs. In [7, 6] it is shown that the question whether a probabilistic program almost surely terminates can be reduced to Problem (1). Using binary search one can find the "critical" value of the parameter for which the program may not terminate any more.

Etessami and Yannakakis show in [7] that Problem (1) can be solved in polynomial time by a reduction to (exact) Linear Programming (LP), which is not known to be strongly polynomial. Our first result reduces Problem (1) to solving a system of linear equations, resulting in a strongly polynomial algorithm for Problem (1). The Maple library offers exact arithmetic solvers for LP and systems of linear equations, which we use to test the performance of our new algorithm. In the neutron branching process discussed above we obtain speed-ups of about one order of magnitude with respect to LP.

The second result of the paper is, to the best of our knowledge, the first practical algorithm for Problem (2). Lower bounds for μ_f can be computed using Newton's method for approximating a root of the function $f(\overline{X}) - \overline{X}$. This has recently been investigated in detail [7, 10, 5]. However, Newton's method faces considerable numerical problems. Experiments show that naive use of exact arithmetic is inefficient, while floating-point computation leads to false results even for very small systems. For instance, the PReMo tool [12], which implements Newton's method with floating-point arithmetic for efficiency, reports $\mu_f \geq \overline{1}$ for a PSP with only 7 variables and small coefficients, although $\mu_f < \overline{1}$ is the case (see Section 3.1).

Our algorithm produces a sequence of guaranteed lower and upper bounds, both of which converge linearly to μ_f. Linear convergence means that, loosely speaking, the number of accurate bits of the bound is a linear function of the position of the bound in the sequence. The algorithm is based on the following idea. Newton's method is an iterative procedure that, given a current lower bound $\mathbf{lb}$ on μ_f, applies a certain operator $\mathcal{N}$ to it, yielding a new, more precise lower bound $\mathcal{N}(\mathbf{lb})$. Instead of computing $\mathcal{N}(\mathbf{lb})$ using exact arithmetic, our algorithm computes *two* consecutive Newton steps, i.e., $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$, using *inexact* arithmetic. Then it checks if the result satisfies a carefully chosen condition. If so, the result is taken as the next lower bound. If not, then the precision is increased, and the computation redone. The condition is eventually satisfied, assuming the results of computing with increased precision converge to the exact result. Usually, the repeated inexact computation is much faster than the exact one. At the same time, a careful (and rather delicate) analysis shows that the sequence of lower bounds converges linearly to μ_f.

Computing *upper* bounds is harder, and seemingly has not been considered in the literature before. Similarly to the case of lower bounds, we apply f twice to $\mathbf{ub}$, i.e., we compute $f(f(\mathbf{ub}))$ with increasing precision until a condition holds. The sequence so obtained may not even converge to μ_f. So we need to introduce a further operation, after which we can then prove linear convergence.

We test our algorithm on the neutron branching process. The time needed to obtain lower and upper bounds on the probability of no explosion with $\epsilon = 0.0001$ lies below the time needed to check, using exact LP, whether this probability is 1 or smaller than one. That is, in this case study our algorithm is faster, and provides more information.

The rest of the paper is structured as follows. We give preliminary definitions and facts in Section 2. Sections 3 and 4 present our algorithms for solving Problems (1) and (2), and report on their performance on some case studies. Section 5 contains our conclusions. The full version of the paper, including all proofs, can be found in [4].

2. Preliminaries

Vectors and matrices. We use bold letters for designating (column) vectors, e.g. $\mathbf{v} \in \mathbb{R}^n$. We write $\overline{s}$ with $s \in \mathbb{R}$ for the vector $(s, \ldots, s)^\top \in \mathbb{R}^n$ (where $^\top$ indicates transpose), if the dimension n is clear from the context. The i-th component of $\mathbf{v} \in \mathbb{R}^n$ will be denoted by $\mathbf{v}_i$. We write $\mathbf{x} = \mathbf{y}$ (resp. $\mathbf{x} \leq \mathbf{y}$ resp. $\mathbf{x} \prec \mathbf{y}$) if $\mathbf{x}_i = \mathbf{y}_i$ (resp. $\mathbf{x}_i \leq \mathbf{y}_i$ resp. $\mathbf{x}_i < \mathbf{y}_i$) holds for all $i \in \{1, \ldots, n\}$. By $\mathbf{x} < \mathbf{y}$ we mean $\mathbf{x} \leq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$. By $\mathbb{R}^{m \times n}$ we denote the set of real matrices with m rows and n columns. We write Id for the identity matrix. For a square matrix A, we denote by $\rho(A)$ the *spectral radius* of A, i.e., the maximum of the absolute values of the eigenvalues. A matrix is *nonnegative* if all its entries are nonnegative. A nonnegative matrix $A \in \mathbb{R}^{n \times n}$ is *irreducible* if for every $k, l \in \{1, \ldots, n\}$ there exists an $i \in \mathbb{N}$ so that $(A^i)_{kl} \neq 0$.

Probabilistic Systems of Polynomials. We investigate equation systems of the form

$$X_1 = f_1(X_1, \ldots, X_n) \quad \ldots \quad X_n = f_n(X_1, \ldots, X_n),$$

where the f_i are polynomials in the variables $X_1, \ldots, X_n$ with positive real coefficients, and for every polynomial f_i the sum of its coefficients is *at most* 1. The vector $f := (f_1, \ldots, f_n)^\top$ is called a *probabilistic system of polynomials* (PSP for short) and is identified with its induced function $f : \mathbb{R}^n \to \mathbb{R}^n$. If $X_1, \ldots, X_n$ are the formal variables of f, we define $\overline{X} := (X_1, \ldots, X_n)^\top$ and Var$(f) := \{X_1, \ldots, X_n\}$. We assume that f is represented as a list of polynomials, and each polynomial is a list of its monomials. If $S \subseteq \{X_1, \ldots, X_n\}$, then f_S denotes the result of removing the polynomial $f_i(X_1, \ldots, X_n)$ from f for every $x_i \notin S$; further, given $\mathbf{x} \in \mathbb{R}^n$ and $B \in \mathbb{R}^{n \times n}$, we denote by $\mathbf{x}_S$ and B_{SS} the vector and the matrix obtained from $\mathbf{x}$ and B by removing the entries with indices i such that $X_i \notin S$. The coefficients are represented as fractions of positive integers. The *size* of f is the size of that representation. The *degree* of f is the maximum of the degrees of $f_1, \ldots, f_n$. PSPs of degree 0 (resp. 1 resp. >1) are called *constant* (resp. *linear* resp. *superlinear*). PSPs f where the degree of each f_i is at least 2 are called *purely superlinear*. We write f' for the *Jacobian* of f, i.e., the matrix of first partial derivatives of f.

Given a PSP f, a variable X_i *depends directly* on a variable X_j if X_j "occurs" in f_i, more formally if $\frac{\partial f_i}{\partial X_j}$ is not the constant 0. A variable X_i *depends* on X_j if X_i depends directly on X_j

or there is a variable X_k such that X_i depends directly on X_k and X_k depends on X_j. We often consider the *strongly connected components* (or SCCs for short) of the dependence relation. The SCCs of a PSP can be computed in linear time using e.g. Tarjan's algorithm. An SCC S of a PSP f is *constant* resp. *linear* resp. *superlinear* resp. *purely superlinear* if the PSP $\tilde{f}$ has the respective property, where $\tilde{f}$ is obtained by restricting f to the S-components and replacing all variables not in S by the constant 1. A PSP is an *scPSP* if it is not constant and consists of only one SCC. Notice that a PSP f is an scPSP if and only if $f'(\overline{1})$ is irreducible.

A fixed point of a PSP f is a vector $\mathbf{x} \geq \overline{0}$ with $f(\mathbf{x}) = \mathbf{x}$. By Kleene's theorem, there exists a least fixed point μ_f of f, i.e., $\mu_f \leq \mathbf{x}$ holds for every fixed point $\mathbf{x}$. Moreover, the sequence $\overline{0}, f(\overline{0}), f(f(\overline{0})), \ldots$ converges to μ_f. Vectors $\mathbf{x}$ with $\mathbf{x} \leq f(\mathbf{x})$ (resp. $\mathbf{x} \geq f(\mathbf{x})$) are called *pre-fixed* (resp. *post-fixed*) points. Notice that the vector $\overline{1}$ is always a post-fixed point of a PSP f, due to our assumption on the coefficients of a PSP. By Knaster-Tarski's theorem, μ_f is the least post-fixed point, so we always have $\overline{0} \leq \mu_f \leq \overline{1}$. It is easy to detect and remove all components i with $(\mu_f)_i = 0$ by a simple round-robin method (see e.g. [5]), which needs linear time in the size of f. We therefore assume in the following that $\mu_f \succ \overline{0}$.

3. An algorithm for consistency of PSPs

Recall that for applications like the neutron branching process it is crucial to know exactly whether $\mu_f = \overline{1}$ holds. We say a PSP f is *consistent* if $\mu_f = \overline{1}$; otherwise it is *inconsistent*. Similarly, we call a component i consistent if $(\mu_f)_i = 1$. We present a new algorithm for the consistency problem, i.e., the problem to check a PSP for consistency.

It was proved in [7] that consistency is checkable in polynomial time by reduction to Linear Programming (LP). We first observe that consistency of general PSPs can be reduced to consistency of scPSPs by computing the DAG of SCCs, and checking consistency SCC-wise [7]: Take any bottom SCC S, and check the consistency of f_S. (Notice that f_S is either constant or an scPSP; if constant, f_S is consistent iff $f_S = 1$, if an scPSP, we can check its consistency by assumption.) If f_S is inconsistent, then so is f, and we are done. If f_S is consistent, then we remove every f_i from f such that $x_i \in S$, replace all variables of S in the remaining polynomials by the constant 1, and iterate (choose a new bottom SCC, etc.). Note that this algorithm processes each polynomial at most once, as every variable belongs to exactly one SCC.

It remains to reduce the consistency problem for scPSPs to LP. The first step is:

Proposition 3.1. [9, 7] *An scPSP f is consistent iff $\rho(f'(\overline{1})) \leq 1$ (i.e., iff the spectral radius of the Jacobi matrix f' evaluated at the vector $\overline{1}$ is at most 1).*

The second step consists of observing that the matrix $f'(\overline{1})$ of an scPSP f is irreducible and nonnegative. It is shown in [7] that $\rho(A) \leq 1$ holds for an irreducible and nonnegative matrix A iff the system of inequalities

$$A\mathbf{x} \geq \mathbf{x} + \overline{1}\,, \mathbf{x} \geq \overline{0} \tag{3.1}$$

is infeasible. However, no strongly polynomial algorithm for LP is known, and we are not aware that (3.1) falls within any subclass solvable in strongly polynomial time [8].

We provide a very simple, strongly polynomial time algorithm to check whether $\rho(f'(\overline{1})) \leq 1$ holds. We need some results from Perron-Frobenius theory (see e.g. [3]).

Lemma 3.2. *Let $A \in \mathbb{R}^{n \times n}$ be nonnegative and irreducible.*

(1) *$\rho(A)$ is a* simple *eigenvalue of A.*
(2) *There exists an eigenvector $\mathbf{v} \succ \overline{0}$ with $\rho(A)$ as eigenvalue.*

(3) *Every eigenvector* $\mathbf{v} \succ \overline{0}$ *has* $\rho(A)$ *as eigenvalue.*
(4) *For all* $\alpha, \beta \in \mathbb{R} \setminus \{0\}$ *and* $\mathbf{v} > \overline{0}$*: if* $\alpha\mathbf{v} < A\mathbf{v} < \beta\mathbf{v}$*, then* $\alpha < \rho(A) < \beta$*.*

The following lemma is the key to the algorithm:

Lemma 3.3. *Let* $A \in \mathbb{R}^{n\times n}$ *be nonnegative and irreducible.*

(a) *Assume there is* $\mathbf{v} \in \mathbb{R}^n \setminus \{\overline{0}\}$ *such that* $(Id - A)\mathbf{v} = \overline{0}$*. Then* $\rho(A) \leq 1$ *iff* $\mathbf{v} \succ \overline{0}$ *or* $\mathbf{v} \prec \overline{0}$*.*
(b) *Assume* $\mathbf{v} = \overline{0}$ *is the only solution of* $(Id - A)\mathbf{v} = \overline{0}$*. Then there exists a unique* $\mathbf{x} \in \mathbb{R}^n$ *such that* $(Id - A)\mathbf{x} = \overline{1}$*, and* $\rho(A) \leq 1$ *iff* $\mathbf{x} \geq \overline{1}$ *and* $A\mathbf{x} < \mathbf{x}$*.*

Proof.

(a) From $(Id - A)\mathbf{v} = \overline{0}$ it follows $A\mathbf{v} = \mathbf{v}$. We see that $\mathbf{v}$ is an eigenvector of A with eigenvalue 1. So $\rho(A) \geq 1$.
($\Leftarrow$): As both $\mathbf{v}$ and $-\mathbf{v}$ are eigenvectors of A with eigenvalue 1, we can assume w.l.o.g. that $\mathbf{v} \succ \overline{0}$. By Lemma 3.2(3), $\rho(A)$ is the eigenvalue of $\mathbf{v}$, and so $\rho(A) = 1$.
($\Rightarrow$): Since $\rho(A) \leq 1$ and $\rho(A) \geq 1$, it follows that $\rho(A) = 1$. By Lemma 3.2(1) and (2), the eigenspace of the eigenvalue 1 is one-dimensional and contains a vector $\mathbf{x} \succ \overline{0}$. So $\mathbf{v} = \alpha \cdot \mathbf{x}$ for some $\alpha \in \mathbb{R}, \alpha \neq 0$. If $\alpha > 0$, we have $\mathbf{v} \succ \overline{0}$, otherwise $\mathbf{v} \prec \overline{0}$.
(b) With the assumption and basic facts from linear algebra it follows that $(Id - A)$ has full rank and therefore $(Id - A)\mathbf{x} = \overline{1}$ has a unique solution $\mathbf{x}$. We still have to prove the second part of the conjunction:
($\Leftarrow$): Follows directly from Lemma 3.2(4).
($\Rightarrow$): Let $\rho(A) \leq 1$. Assume for a contradiction that $\rho(A) = 1$. Then, by Lemma 3.2(1), the matrix A would have an eigenvector $\mathbf{v} \neq \overline{0}$ with eigenvalue 1, so $(Id - A)\mathbf{v} = \overline{0}$, contradicting the assumption. So we have, in fact, $\rho(A) < 1$. By standard matrix facts (see e.g. [3]), this implies that $(Id - A)^{-1} = A^* = \sum_{i=0}^{\infty} A^i$ exists, and so we have $\mathbf{x} = (Id - A)^{-1}\overline{1} = A^*\overline{1} \geq \overline{1}$. Furthermore, $A\mathbf{x} = \sum_{i=1}^{\infty} A^i\overline{1} < \sum_{i=0}^{\infty} A^i\overline{1} = \mathbf{x}$. ∎

In order to check whether $\rho(A) \leq 1$, we first solve the system $(Id - A)\mathbf{v} = \overline{0}$ using Gaussian elimination. If we find a vector $\mathbf{v} \neq \overline{0}$ such that $(Id - A)\mathbf{v} = \overline{0}$, we apply Lemma 3.3(a). If $\mathbf{v} = \overline{0}$ is the only solution of $(Id - A)\mathbf{v} = \overline{0}$, we solve $(Id - A)\mathbf{v} = \overline{1}$ using Gaussian elimination again, and apply Lemma 3.3(b). Since Gaussian elimination of a rational n-dimensional linear equation system can be carried out in strongly polynomial time using $O(n^3)$ arithmetic operations (see e.g. [8]), we obtain:

Proposition 3.4. *Given a nonnegative irreducible matrix* $A \in \mathbb{R}^{n\times n}$*, one can decide in strongly polynomial time, using* $O(n^3)$ *arithmetic operations, whether* $\rho(A) \leq 1$*.*

Combining Propositions 3.1 and 3.4 directly yields an algorithm for checking the consistency of scPSPs. Extending it to multiple SCCs as above, we get:

Theorem 3.5. *Let* $f(X_1, \ldots, X_n)$ *be a PSP. There is a strongly polynomial time algorithm that uses* $O(n^3)$ *arithmetic operations and determines the consistency of* f*.*

3.1. Case study: A family of "almost consistent" PSPs

In this section, we illustrate some issues faced by algorithms that solve the consistency problem. Consider the following family $h^{(n)}$ of scPSPs, $n \geq 2$:

$$h^{(n)} = \left(\ 0.5X_1^2 + 0.1X_n^2 + 0.4\ ,\ 0.01X_1^2 + 0.5X_2 + 0.49\ ,\ \ldots\ , 0.01X_{n-1}^2 + 0.5X_n + 0.49\ \right)^\top .$$

	$n = 25$	$n = 100$	$n = 200$	$n = 400$	$n = 600$	$n = 1000$
Exact LP	< 1 sec	2 sec	8 sec	67 sec	208 sec	> 2h
Our algorithm	< 1 sec	< 1 sec	1 sec	4 sec	10 sec	29 sec

Table 1: Consistency checks for $h^{(n)}$-systems: Runtimes of different approaches.

It is not hard to show that $h^{(n)}(\mathbf{p}) \prec \mathbf{p}$ holds for $\mathbf{p} = (1 - 0.02^n, \ldots, 1 - 0.02^{2n-1})^\top$, so we have $\mu_{h^{(n)}} \prec \overline{1}$ by Proposition 4.4, i.e., the $h^{(n)}$ are inconsistent.

The tool PReMo [12] relies on Java's floating-point arithmetic to compute approximations of the least fixed point of a PSP. We invoked PReMo for computing approximants of $\mu_{h^{(n)}}$ for different values of n between 5 and 100. Due to its fixed precision, PReMo's approximations for $\mu_{h^{(n)}}$ are ≥ 1 in all components if $n \geq 7$. This might lead to the wrong conclusion that $h^{(n)}$ is consistent.

Recall that the consistency problem can be solved by checking the feasibility of the system (3.1) with $A = f'(\overline{1})$. We checked it with lp_solve, a well-known LP tool using hardware floating-point arithmetic. The tool wrongly states that (3.1) has no solution for $h^{(n)}$-systems with $n > 10$. This is due to the fact that the solutions cannot be represented adequately using machine number precision.[2] Finally, we also checked feasibility with Maple's Simplex package, which uses exact arithmetic, and compared its performance with the implementation, also in Maple, of our consistency algorithm. Table 1 shows the results. Our algorithm clearly outperforms the LP approach. For more experiments see Section 4.3.

4. Approximating μ_f with inexact arithmetic

It is shown in [7] that μ_f may not be representable by roots, so one can only approximate μ_f. In this section we present an algorithm that computes two sequences, $(\mathbf{lb}^{(i)})_i$ and $(\mathbf{ub}^{(i)})_i$, such that $\mathbf{lb}^{(i)} \leq \mu_f \leq \mathbf{ub}^{(i)}$ and $\lim_{i\to\infty} \mathbf{ub}^{(i)} - \mathbf{lb}^{(i)} = \overline{0}$. In words: $\mathbf{lb}^{(i)}$ and $\mathbf{ub}^{(i)}$ are lower and upper bounds on μ_f, respectively, and the sequences converge to μ_f. Moreover, they converge linearly, meaning that the *number of accurate bits* of $\mathbf{lb}^{(i)}$ and $\mathbf{ub}^{(i)}$ are linear functions of i. (The number of accurate bits of a vector $\mathbf{x}$ is defined as the greatest number k such that $|(\mu_f - \mathbf{x})_j|/|(\mu_f)_j| \leq 2^{-k}$ holds for all $j \in \{1, \ldots, n\}$.) These properties are guaranteed even though our algorithm uses inexact arithmetic: Our algorithm detects numerical problems due to rounding errors, recovers from them, and increases the precision of the arithmetic as needed. Increasing the precision dynamically is, e.g., supported by the GMP library [1].

Let us make precise what we mean by increasing the precision. Consider an elementary operation g, like multiplication, subtraction, etc., that operates on two input numbers x and y. We can *compute* $g(x, y)$ *with increasing precision* if there is a procedure that on input x, y outputs a sequence $g^{(1)}(x, y), g^{(2)}(x, y), \ldots$ that converges to $g(x, y)$. Note that there are no requirements on the convergence speed of this procedure — in particular, we do not require that there is an i with $g^{(i)}(x, y) = g(x, y)$. This procedure, which we assume exists, allows to implement *floating assignments* of the form

$$z \leftsquigarrow g(x, y) \textbf{ such that } \phi(z)$$

with the following semantics: z is assigned the value $g^{(i)}(x, y)$, where $i \geq 1$ is the smallest index such that $\phi(g^{(i)}(x, y))$ holds. We say that the assignment is *valid* if $\phi(g(x, y))$ holds and ϕ involves

[2] The mentioned problems of PReMo and lp_solve are not due to the fact that the coefficients of $h^{(n)}$ cannot be properly represented using basis 2: The problems persist if one replaces the coefficients of $h^{(n)}$ by similar numbers exactly representable by machine numbers.

only continuous functions and strict inequalities. Our assumption on the arithmetic guarantees that (the computation underlying) a valid floating assignment terminates. As “syntactic sugar”, more complex operations (e.g., linear equation solving) are also allowed in floating assignments, because they can be decomposed into elementary operations.

We feel that any implementation of arbitrary precision arithmetic should satisfy our requirement that the computed values converge to the exact result. For instance, the documentation of the GMP library [1] states: “Each function is defined to calculate with ‘infinite precision’ followed by a truncation to the destination precision, but of course the work done is only what’s needed to determine a result under that definition.”

To approximate the least fixed point of a PSP, we first transform it into a certain normal form. A purely superlinear PSP f is called *perfectly superlinear* if every variable depends directly on itself and every superlinear SCC is purely superlinear. The following proposition states that any PSP f can be made perfectly superlinear.

Proposition 4.1. *Let f be a PSP of size s. We can compute in time $O(n \cdot s)$ a perfectly superlinear PSP $\tilde{f}$ with $\mathit{Var}(\tilde{f}) = \mathit{Var}(f) \cup \{\tilde{X}\}$ of size $O(n \cdot s)$ such that $\mu_f = (\mu_{\tilde{f}})_{\mathit{Var}(f)}$.*

4.1. The algorithm

The algorithm receives as input a perfectly superlinear PSP f and an error bound $\epsilon > 0$, and returns vectors $\mathbf{lb}, \mathbf{ub}$ such that $\mathbf{lb} \le \mu_f \le \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \le \overline{\epsilon}$. A first initialization step requires to compute a vector $\mathbf{x}$ with $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x})$, i.e., a “strict” pre-fixed point. This is done in Section 4.1.1. The algorithm itself is described in Section 4.1.2.

4.1.1. *Computing a strict pre-fixed point.* Algorithm 1 computes a strict pre-fixed point:

```
Algorithm 1: Procedure computeStrictPrefix
Input: perfectly superlinear PSP $f$
Output: $\mathbf{x}$ with $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x}) \prec \overline{1}$
$\mathbf{x} \leftarrow \overline{0}$;
while $\overline{0} \not\prec \mathbf{x}$ do
    $Z \leftarrow \{i \mid 1 \le i \le n, f_i(\mathbf{x}) = 0\}$;
    $P \leftarrow \{i \mid 1 \le i \le n, f_i(\mathbf{x}) > 0\}$;
    $\mathbf{y}_Z \leftarrow \overline{0}$;
    $\mathbf{y}_P \leftrightsquigarrow f_P(\mathbf{x})$ such that $\overline{0} \prec \mathbf{y}_P \prec f_P(\mathbf{y}) \prec \overline{1}$;
    $\mathbf{x} \leftarrow \mathbf{y}$;
```

Proposition 4.2. *Algorithm 1 is correct and terminates after at most n iterations.*

The reader may wonder why Algorithm 1 uses a floating assignment $\mathbf{y}_P \leftrightsquigarrow f_P(\mathbf{x})$, given that it must also perform exact comparisons to obtain the sets Z and P and to decide exactly whether $\mathbf{y}_P \prec f_P(\mathbf{y})$ holds in the **such that** clause of the floating assignment. The reason is that, while we perform such operations exactly, we do not want to use the *result* of exact computations as input for other computations, as this easily leads to an explosion in the required precision. For instance, the size of the exact result of $f_P(\mathbf{y})$ may be larger than the size of $\mathbf{y}$, while an approximation of smaller size may already satisfy the **such that** clause. In order to emphasize this, we *never* store the result of an exact numerical computation in a variable.

4.1.2. *Computing lower and upper bounds.* Algorithm 1 uses Kleene iteration $\overline{0}, f(\overline{0}), f(f(\overline{0})), \ldots$ to compute a strict pre-fixed point. One could, in principle, use the same scheme to compute lower bounds of μ_f, as this sequence converges to μ_f from below by Kleene's theorem. However, convergence of Kleene iteration is generally slow. It is shown in [7] that for the 1-dimensional PSP f with $f(X) = 0.5X^2 + 0.5$ we have $\mu_f = 1$, and the i-th Kleene approximant $\boldsymbol{\kappa}^{(i)}$ satisfies $\boldsymbol{\kappa}^{(i)} \leq 1 - \frac{1}{i}$. Hence, Kleene iteration may converge only logarithmically, i.e., the number of accurate bits is a logarithmic function of the number of iterations.

In [7] it was suggested to use Newton's method for faster convergence. In order to see how Newton's method can be used, observe that instead of computing μ_f, one can equivalently compute the least nonnegative zero of $f(\overline{X}) - \overline{X}$. Given an approximant $\mathbf{x}$ of μ_f, Newton's method first computes $g^{(\mathbf{x})}(\overline{X})$, the first-order linearization of f at the point $\mathbf{x}$:

$$g^{(\mathbf{x})}(\overline{X}) = f(\mathbf{x}) + f'(\mathbf{x})(\overline{X} - \mathbf{x})$$

The next Newton approximant $\mathbf{y}$ is obtained by solving $\overline{X} = g^{(\mathbf{x})}(\overline{X})$, i.e.,

$$\mathbf{y} = \mathbf{x} + (Id - f'(\mathbf{x}))^{-1}(f(\mathbf{x}) - \mathbf{x}) \ .$$

We write $\mathcal{N}_f(\mathbf{x}) := \mathbf{x} + (Id - f'(\mathbf{x}))^{-1}(f(\mathbf{x}) - \mathbf{x})$, and usually drop the subscript of $\mathcal{N}_f$. If $\boldsymbol{\nu}^{(0)} \leq \mu_f$ is any pre-fixed point of f, for instance $\boldsymbol{\nu}^{(0)} = \overline{0}$, we can define a *Newton sequence* $(\boldsymbol{\nu}^{(i)})_i$ by setting $\boldsymbol{\nu}^{(i+1)} = \mathcal{N}(\boldsymbol{\nu}^{(i)})$ for $i \geq 0$. It has been shown in [7, 10, 5] that Newton sequences converge at least linearly to μ_f. Moreover, we have $\overline{0} \leq \boldsymbol{\nu}^{(i)} \leq f(\boldsymbol{\nu}^{(i)}) \leq \mu_f$ for all i.

These facts were shown only for Newton sequences that are computed exactly, i.e., without rounding errors. Unfortunately, Newton approximants are hard to compute exactly: Since each iteration requires to solve a linear equation system whose coefficients depend on the results of the previous iteration, the size of the Newton approximants easily explodes. Therefore, we wish to use inexact arithmetic, but without losing the good properties of Newton's method (reliable lower bounds, linear convergence).

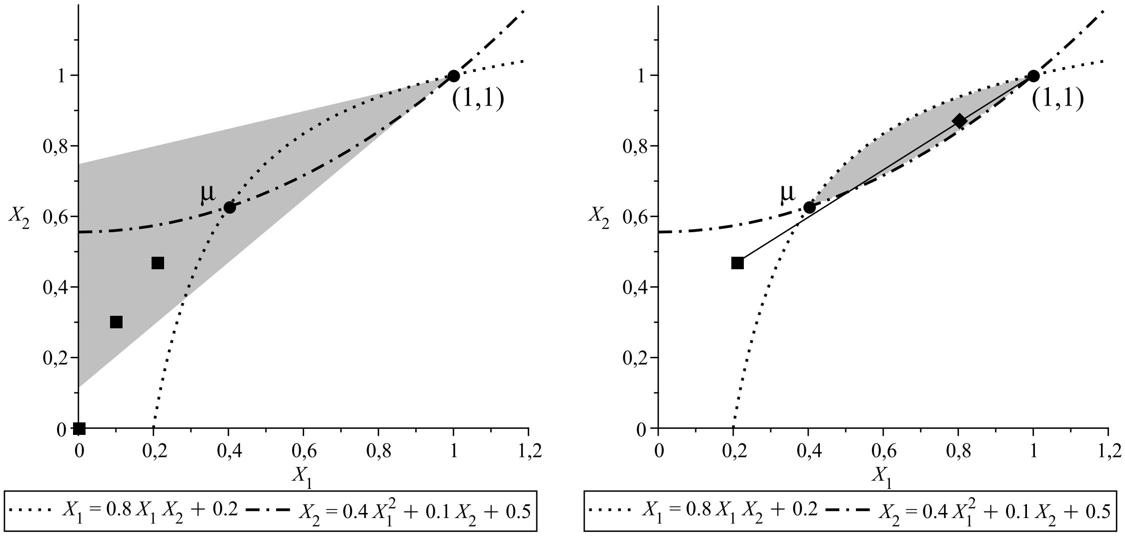
Algorithm 2 accomplishes these goals, and additionally computes post-fixed points $\mathbf{ub}$ of f, which are upper bounds on μ_f. Let us describe the algorithm in some detail. The lower bounds are stored in the variable $\mathbf{lb}$. The first value of $\mathbf{lb}$ is not simply $\overline{0}$, but is computed by `computeStrictPrefix`(f), in order to guarantee the validity of the following floating assignments. We use Newton's method for improving the lower bounds because it converges fast (at least linearly) when performed exactly. In each iteration of the algorithm, *two* Newton steps are performed using inexact arithmetic. The intention is that two inexact Newton steps should improve the lower bound at least as much as one exact Newton step. While this may sound like a vague hope for small rounding errors, it can be rigorously proved thanks to the **such that** clause of the floating assignment in line 4. The proof involves two steps. The first step is to prove that $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$ is a (strict) post-fixed point of the function $g(\overline{X}) = f(\mathbf{lb}) + f'(\mathbf{lb})(\overline{X} - \mathbf{lb})$, i.e., $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$ satisfies the first inequality in the **such that** clause. For the second step, recall that $\mathcal{N}(\mathbf{lb})$ is the least fixed point of g. By Knaster-Tarski's theorem, $\mathcal{N}(\mathbf{lb})$ is actually the least post-fixed point of g. So, our value $\mathbf{x}$, the inexact version of $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$, satisfies $\mathbf{x} \geq \mathcal{N}(\mathbf{lb})$, and hence two inexact Newton steps are in fact at least as "fast" as one exact Newton step. Thus, the $\mathbf{lb}$ converge linearly to μ_f.

The upper bounds $\mathbf{ub}$ are post-fixed points, i.e., $f(\mathbf{ub}) \leq \mathbf{ub}$ is an invariant of the algorithm. The algorithm computes the sets Z and P so that inexact arithmetic is only applied to the components i with $f_i(\mathbf{ub}) < 1$. In the P-components, the function f is applied to $\mathbf{ub}$ in order to improve the upper bound. In fact, f is applied twice in line 9, analogously to applying $\mathcal{N}$ twice in line 4. Here, the **such that** clause makes sure that the progress towards μ_f is at least as fast as the progress of one exact application of f would be. One can show that this leads to linear convergence to μ_f.

Algorithm 2: Procedure `calcBounds`

```
   Input: perfectly superlinear PSP $f$, error bound $\epsilon > 0$
   Output: vectors $\mathbf{lb}, \mathbf{ub}$ such that $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$
1  $\mathbf{lb} \leftarrow$ computeStrictPrefix$(f)$;
2  $\mathbf{ub} \leftarrow \overline{1}$;
3  while $\mathbf{ub} - \mathbf{lb} \not\leq \overline{\epsilon}$ do
4      $\mathbf{x} \leftsquigarrow \mathcal{N}(\mathcal{N}(\mathbf{lb}))$ such that $f(\mathbf{lb}) + f'(\mathbf{lb})(\mathbf{x} - \mathbf{lb}) \prec \mathbf{x} \prec f(\mathbf{x}) \prec \overline{1}$;
5      $\mathbf{lb} \leftarrow \mathbf{x}$;
6      $Z \leftarrow \{i \mid 1 \leq i \leq n, f_i(\mathbf{ub}) = 1\}$;
7      $P \leftarrow \{i \mid 1 \leq i \leq n, f_i(\mathbf{ub}) < 1\}$;
8      $\mathbf{y}_Z \leftarrow \overline{1}$;
9      $\mathbf{y}_P \leftsquigarrow f_P(f(\mathbf{ub}))$ such that $f_P(\mathbf{y}) \prec \mathbf{y}_P \prec f_P(\mathbf{ub})$;
10     forall superlinear SCCs $S$ of $f$ with $\mathbf{y}_S = \overline{1}$ do
11         $\mathbf{t} \leftarrow \overline{1} - \mathbf{lb}_S$;
12         if $f'_{SS}(\overline{1})\mathbf{t} \succ \mathbf{t}$ then
13             $\mathbf{y}_S \leftsquigarrow \overline{1} - \min\left\{1, \frac{\min_{i \in S}(f'_{SS}(\overline{1})\mathbf{t} - \mathbf{t})_i}{2 \cdot \max_{i \in S}(f_S(\overline{2}))_i}\right\} \cdot \mathbf{t}$ such that $f_S(\mathbf{y}) \prec \mathbf{y}_S \prec \overline{1}$;
14     $\mathbf{ub} \leftarrow \mathbf{y}$;
```

The rest of the algorithm (lines 10-13) deals with the problem that, given a post-fixed $\mathbf{ub}$, the sequence $\mathbf{ub}, f(\mathbf{ub}), f(f(\mathbf{ub})), \ldots$ does not necessarily converge to μ_f. For instance, if $f(X) = 0.75X^2 + 0.25$, then $\mu_f = 1/3$, but $1 = f(1) = f(f(1)) = \cdots$. Therefore, the if-statement of Algorithm 2 allows to improve the upper bound from $\overline{1}$ to a post-fixed point less than $\overline{1}$, by exploiting the lower bounds $\mathbf{lb}$. This is illustrated in Figure 1 for a 2-dimensional scPSP f. The



(a) (b)

Figure 1: Computation of a post-fixed point less than $\overline{1}$.

dotted lines indicate the curve of the points (X_1, X_2) satisfying $X_1 = 0.8X_1X_2 + 0.2$ and $X_2 = 0.4X_1^2 + 0.1X_2 + 0.5$. Notice that $\mu_f \prec \overline{1} = f(\overline{1})$. In Figure 1 (a) the shaded area consists of those points $\mathbf{lb}$ where $f'(\overline{1})(\overline{1} - \mathbf{lb}) \succ \overline{1} - \mathbf{lb}$ holds, i.e., the condition of line 12. One can show that μ_f must lie in the shaded area, so by continuity, any sequence converging to μ_f, in particular the sequence of lower bounds $\mathbf{lb}$, finally reaches the shaded area. In Figure 1 (a) this is indicated by the points with the square shape. Figure 1 (b) shows how to exploit such a point $\mathbf{lb}$ to compute a post-fixed point $\mathbf{ub} \prec \overline{1}$ (post-fixed points are shaded in Figure 1 (b)): The post-fixed point $\mathbf{ub}$ (diamond shape) is obtained by starting at $\overline{1}$ and moving a little bit along the straight line between $\overline{1}$ and $\mathbf{lb}$, cf. line 13. The sequence $\mathbf{ub}, f(\mathbf{ub}), f(f(\mathbf{ub})), \ldots$ now converges linearly to μ_f.

Theorem 4.3. *Algorithm 2 terminates and computes vectors* $\mathbf{lb}, \mathbf{ub}$ *such that* $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ *and* $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$. *Moreover, the sequences of lower and upper bounds computed by the algorithm both converge linearly to* μ_f.

Notice that Theorem 4.3 is about the convergence speed of the approximants, not about the time needed to compute them. To analyse the computation time, one would need stronger requirements on how floating assignments are performed.

The lower and upper bounds computed by Algorithm 2 have a special feature: they satisfy $\mathbf{lb} \prec f(\mathbf{lb})$ and $\mathbf{ub} \geq f(\mathbf{ub})$. The following proposition guarantees that such points are in fact lower and upper bounds.

Proposition 4.4. *Let* f *be a perfectly superlinear PSP. Let* $\overline{0} \leq \mathbf{x} \leq \overline{1}$. *If* $\mathbf{x} \prec f(\mathbf{x})$, *then* $\mathbf{x} \prec \mu_f$. *If* $\mathbf{x} \geq f(\mathbf{x})$, *then* $\mathbf{x} \geq \mu_f$.

So a user of Algorithm 2 can immediately verify that the computed bounds are correct. To summarize, Algorithm 2 computes provably and even verifiably correct lower and upper bounds, although exact computation is restricted to detecting numerical problems. See Section 4.3 for experiments.

4.2. Proving consistency using the inexact algorithm

In Section 3 we presented a simple and efficient algorithm to check the consistency of a PSP. Algorithm 2 is aimed at approximating μ_f, but note that it can also prove the inconsistency of a PSP: when the algorithm sets $\mathbf{ub}_i < 1$, we know $(\mu_f)_i < 1$. This raises the question whether Algorithm 2 can also be used for proving consistency. The answer is yes, and the procedure is based on the following proposition.

Proposition 4.5. *Let* f *be an scPSP. Let* $\mathbf{t} \succ \overline{0}$ *be a vector with* $f'(\overline{1})\mathbf{t} \leq \mathbf{t}$. *Then* f *is consistent.*

Proposition 4.5 can be used to identify consistent components.

Use Algorithm 2 with some (small) ϵ to compute $\mathbf{ub}$ and $\mathbf{lb}$. Take any bottom SCC S.

- If $f'(\overline{1})(\overline{1} - \mathbf{lb}_S) \leq \overline{1} - \mathbf{lb}_S$, mark all variables in S as consistent and remove the S-components from f. In the remaining components, replace all variables in S with 1.
- Otherwise, remove S and all other variables that depend on S from f.

Repeat with the new bottom SCC until all SCCs are processed.

There is no guarantee that this method detects all i with $(\mu_f)_i = 1$.

D	2			3			6			10		
n	20	50	100	20	50	100	20	50	100	20	50	100
inconsistent (yes/no)	n	n	n	y	y	y	y	y	y	y	y	y
Cons. check (Alg. Sec. 3)	< 1	< 1	2	< 1	< 1	2	< 1	< 1	2	< 1	< 1	2
Cons. check (exact LP)	< 1	20	258	< 1	22	124	< 1	16	168	< 1	37	222
Approx. Q_D ($\epsilon = 10^{-3}$)	< 1	< 1	4	2	8	32	1	5	21	1	4	17
Approx. Q_D ($\epsilon = 10^{-4}$)	< 1	< 1	4	2	8	34	2	7	28	1	6	23

Table 2: Runtime in seconds of various algorithms on different values of D and n.

4.3. Case study: A neutron branching process

One of the main applications of the theory of branching processes is the modelling of cascade creation of particles in physics. We study a problem described by Harris in [9]. Consider a ball of fissionable radioactive material of radius D. Spontaneous fission of an atom can liberate a neutron, whose collision with another atom can produce further neutrons etc. If D is very small, most neutrons leave the ball without colliding. If D is very large, then nearly all neutrons eventually collide, and the probability that the neutron's progeny never dies is large. A well-known result shows that, loosely speaking, the population of a process that does not go extinct grows exponentially over time with large probability. Therefore, the neutron's progeny never dying out actually means that after a (very) short time all the material is fissioned, which amounts to a nuclear explosion. The task is to compute the largest value of D for which the probability of extinction of a neutron born at the centre of the ball is still 1 (if the probability is 1 at the centre, then it is 1 everywhere). This is often called the critical radius. Notice that, since the number of atoms that undergo spontaneous fission is large (some hundreds per second for the critical radius of plutonium), if the probability of extinction lies only slightly below 1, there is already a large probability of a chain reaction. Assume that a neutron born at distance ξ from the centre leaves the ball without colliding with probability $l(\xi)$, and collides with an atom at distance η from the centre with probability density $R(\xi, \eta)$. Let further $f(x) = \sum_{i \geq 0} p_i x^i$, where p_i is the probability that a collision generates i neutrons. For a neutron's progeny to go extinct, the neutron must either leave the ball without colliding, or collide at some distance η from the centre, but in such a way that the progeny of all generated neutrons goes extinct. So the extinction probability $Q_D(\xi)$ of a neutron born at distance ξ from the centre is given by [9], p. 86:

$$Q_D(\xi) = l(\xi) + \int_0^D R(\xi, \eta) f(Q_D(\eta))\, d\eta$$

Harris takes $f(x) = 0.025 + 0.830x + 0.07x^2 + 0.05x^3 + 0.025x^4$, and gives expressions for both $l(\xi)$ and $R(\xi, \eta)$. By discretizing the interval $[0, D]$ into n segments and replacing the integral by a finite sum we obtain a PSP of dimension $n + 1$ over the variables $\{Q_D(jD/n) \mid 0 \leq j \leq n\}$. Notice that $Q_D(0)$ is the probability that a neutron born in the centre does not cause an explosion.

Results. For our experiments we used three different discretizations $n = 20, 50, 100$. We applied our consistency algorithm from Section 3 and Maple's Simplex to check inconsistency, i.e., to check whether an explosion occurs. The results are given in the first 3 rows of Table 2: Again our algorithm dominates the LP approach, although the polynomials are much denser than in the $h^{(n)}$-systems.

We also implemented Algorithm 2 using Maple for computing lower and upper bounds on $Q_D(0)$ with two different values of the error bound ϵ. The runtime is given in the last two rows. By setting the *Digits* variable in Maple we controlled the precision of Maple's software floating-point numbers for the floating assignments. In all cases starting with the standard value of 10, Algorithm 2 increased *Digits* at most twice by 5, resulting in a maximal *Digits* value of 20. We mention that Algorithm 2 computed an upper bound $\prec \overline{1}$, and thus proved inconsistency, after the first few iterations in all investigated cases, almost as fast as the algorithm from Section 3.

Computing approximations for the critical radius. After computing $Q_D(0)$ for various values of D one can suspect that the critical radius, i.e., the smallest value of D for which $Q_D(0) = 1$, lies somewhere between 2.7 and 3. We combined binary search with the consistency algorithm from Section 3 to determine the critical radius up to an error of 0.01. During the binary search, the algorithm from Section 3 has to analyze PSPs that come closer and closer to the verge of (in)consistency. For the last (and most expensive) binary search step that decreases the interval to 0.01, our algorithm took <1, 1, 3, 8 seconds for $n = 20, 50, 100, 150$, respectively. For $n = 150$, we found the critical radius to be in the interval $[2.981, 2.991]$. Harris [9] estimates 2.9.

5. Conclusions

We have presented a new, simple, and efficient algorithm for checking the consistency of PSPs, which outperforms the previously existing LP-based method. We have also described the first algorithm that computes reliable lower and upper bounds on μ_f. The sequence of bounds converges linearly to μ_f. To achieve these properties without sacrificing efficiency, we use a novel combination of exact and inexact (floating-point) arithmetic. Experiments on PSPs from concrete branching processes confirm the practicality of our approach. The results raise the question whether our combination of exact and inexact arithmetic could be transferred to other computational problems.

Acknowledgments. We thank several anonymous referees for pointing out inaccuracies and helping us clarify certain aspects of the paper. The second author was supported by the DFG Graduiertenkolleg 1480 (PUMA). We also thank Andreas Reuss for proofreading the manuscript.

References

[1] GMP library. http://gmplib.org.

[2] K. B. Athreya and P. E. Ney. *Branching Processes*. Springer, 1972.

[3] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. SIAM, 1994.

[4] J. Esparza, A. Gaiser, and S. Kiefer. Computing least fixed points of probabilistic systems of polynomials. Technical report, Technische Universität München, Institut für Informatik, 2009.

[5] J. Esparza, S. Kiefer, and M. Luttenberger. Convergence thresholds of Newton's method for monotone polynomial equations. In *Proceedings of STACS*, pages 289–300, 2008.

[6] J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. In *LICS 2004*, pages 12–21. IEEE Computer Society, 2004.

[7] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66, 2009.

[8] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1993.

[9] T. E. Harris. *The theory of branching processes*. Springer, Berlin, 1963.

[10] S. Kiefer, M. Luttenberger, and J. Esparza. On the convergence of Newton's method for monotone systems of polynomial equations. In *Proceedings of STOC*, pages 217–226. ACM, 2007.

[11] C. D. Manning and H. Schuetze. *Foundations of Statistical Natural Language Processing*. MIT Press, June 1999.

[12] D. Wojtczak and K. Etessami. PReMo: an analyzer for probabilistic recursive models. In *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 66–71. Springer, 2007.

THE k-IN-A-PATH PROBLEM FOR CLAW-FREE GRAPHS

JIŘÍ FIALA [1] AND MARCIN KAMIŃSKI [2] AND BERNARD LIDICKÝ [1] AND DANIËL PAULUSMA [3]

[1] Charles University, Faculty of Mathematics and Physics,
DIMATIA and Institute for Theoretical Computer Science (ITI)
Malostranské nám. 2/25, 118 00, Prague, Czech Republic
E-mail address: fiala@kam.mff.cuni.cz
E-mail address: bernard@kam.mff.cuni.cz

[2] Computer Science Department, Université Libre de Bruxelles,
Boulevard du Triomphe CP212, B-1050 Brussels, Belgium
E-mail address: marcin.kaminski@ulb.ac.be

[3] Department of Computer Science, University of Durham,
Science Laboratories, South Road,
Durham DH1 3LE, England
E-mail address: daniel.paulusma@durham.ac.uk

Abstract. Testing whether there is an induced path in a graph spanning k given vertices is already NP-complete in general graphs when $k = 3$. We show how to solve this problem in polynomial time on claw-free graphs, when k is not part of the input but an arbitrarily fixed integer.

1. Introduction

Many interesting graph classes are closed under vertex deletion. Every such class can be characterized by a set of forbidden induced subgraphs. One of the best-known examples is the class of perfect graphs. A little over 40 years after Berge's conjecture, Chudnovsky et al. [18] proved that a graph is perfect if and only if it contains neither an *odd hole* (induced cycle of odd length) nor an *odd antihole* (complement of an odd hole). This motivates the research of detecting induced subgraphs such as paths and cycles, which is the topic of this paper. To be more precise, we specify some vertices of a graph called the *terminals* and study the computational complexity of deciding if a graph has an induced subgraph of a certain type containing all the terminals. In particular, we focus on the following problem.

1998 ACM Subject Classification: G.2.2 Graph algorithms, F.2.2 Computations on discrete structures.
Key words and phrases: induced path, claw-free graph, polynomial-time algorithm.
Research supported by the Ministry of Education of the Czech Republic as projects 1M0021620808 and GACR 201/09/0197, by the Royal Society Joint Project Grant JP090172 and by EPSRC as EP/D053633/1.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2469

k-in-a-Path
Instance: a graph G with k terminals.
Question: does there exist an induced path of G containing the k terminals?

Note that in the problem above, k is a fixed integer. Clearly, the problem is polynomially solvable for $k = 2$. Haas and Hoffmann [11] consider the case $k = 3$. After pointing out that this case is NP-complete as a consequence of a result by Fellows [9], they prove W[1]-completeness (where they take as parameter the length of an induced path that is a solution for 3-in-a-Path). Derhy and Picouleau [6] proved that the case $k = 3$ is NP-complete even for graphs with maximum degree at most three.

A natural question is what will happen if we relax the condition of "being contained in an induced path" to "being contained in an induced tree". This leads to the following problem.

k-in-a-Tree
Instance: a graph G with k terminals.
Question: does there exist an induced tree of G containing the k terminals?

As we will see, also this problem has received a lot of attention in the last two years. It is NP-complete if k is part of the input [6]. However, Chudnovsky and Seymour [4] have recently given a deep and complicated polynomial-time algorithm for the case $k = 3$.

Theorem 1.1 ([4])**.** *The* 3-in-a-Tree *problem is solvable in polynomial time.*

The computational complexity of k-in-a-Tree for $k = 4$ is still open. So far, only partial results are known, such as a polynomial-time algorithm for $k = 4$ when the input is triangle-free by Derhy, Picouleau and Trotignon [7]. This result and Theorem 1.1 were extended by Trotignon and Wei [20] who showed that k-in-a-Tree is polynomially solvable for graphs of girth at least k. The authors of [7] also show that it is NP-complete to decide if a graph G contains an induced tree T covering four specified vertices such that T has at most one vertex of degree at least three.

In general, k-in-a-Path and k-in-a-Tree are only equivalent for $k \leq 2$. However, in this paper, we study *claw-free* graphs (graphs with no induced 4-vertex star). Claw-free graphs are a rich and well-studied class containing, e.g., the class of (quasi)-line graphs and the class of complements of triangle-free graphs; see [8] for a survey. Notice that any induced tree in a claw-free graph is in fact an induced path.

Observation 1.2. The k-in-a-Path and k-in-a-Tree problem are equivalent for the class of claw-free graphs.

Motivation. The polynomial-time algorithm for 3-in-a-Tree [4] has already proven to be a powerful tool for several problems. For instance, it is used as a subroutine in polynomial time algorithms for detecting induced thetas and pyramids [4] and several other induced subgraphs [16]. The authors of [12] use it to solve the Parity Path problem in polynomial time for claw-free graphs. (This problem is to test if a graph contains both an odd and even length induced paths between two specified vertices. It is NP-complete in general as shown by Bienstock [1].)

Lévêque et al. [16] use the algorithm of [4] to solve the 2-Induced Cycle problem in polynomial time for graphs not containing an induced path or subdivided claw on some fixed number of vertices. The k-Induced Cycle problem is to test if a graph contains an induced cycle spanning k terminals. In general it is NP-complete already for $k = 2$ [1]. For fixed k, an instance of this problem can be reduced to a polynomial number of instances of the k-Induced Disjoint Paths problem, which we define below. Paths $P_1, \ldots, P_k$ in a graph G are said to be *mutually induced* if for any $1 \leq i < j \leq k$, P_i and P_j have neither common vertices (i.e. $V(P_i) \cap V(P_j) = \emptyset$) nor adjacent vertices (i.e. $uv \notin E$ for any $u \in V(P_i), v \in V(P_j)$).

k-Induced Disjoint Paths
Instance: a graph G with k pairs of terminals (s_i, t_i) for $i = 1, \ldots, k$.
Question: does G contain k mutually induced paths P_i such that P_i connects s_i and t_i for $i = 1, \ldots, k$?

This problem is NP-complete for $k = 2$ [1]. Kawarabayashi and Kobayashi [14] showed that, for any fixed k, the k-Induced Disjoint Paths problem is solvable in linear time on planar graphs and that consequently k-Induced Disjoint Cycle is solvable in polynomial time on this graph class for any fixed k. In [15], Kawarabayashi and Kobayashi improve the latter result by presenting a linear time algorithm for this problem, and even extend the results for both these problems to graphs of bounded genus. As we shall see, we can also solve k-Induced Disjoint Paths and k-Induced Cycle in polynomial time in claw-free graphs. The version of the problem in which any two paths are vertex-disjoint but may have adjacent vertices is called the k-Disjoint Paths problem. For this problem Robertson and Seymour [17] proved the following result.

Theorem 1.3 ([17])**.** *For fixed k, the k-*Disjoint Paths *problem is solvable in polynomial time.*

Our Results and Paper Organization. In Section 2 we define some basic terminology. Section 3 contains our main result: k-in-a-Path is solvable in polynomial time in claw-free graphs for any fixed integer k. This, in fact, follows from a stronger theorem proved in Section 4; the problem is solvable in polynomial time even if the terminals are to appear on the path in a fixed order. A consequence of our result is that the k-Induced Disjoint Paths and k-Induced Cycle problems are polynomially solvable in claw-free graphs for any fixed integer k. In Section 4 we present our polynomial-time algorithm that solves the ordered version of k-in-a-Path. The algorithm first performs "cleaning of the graph". This is an operation introduced in [12]. After cleaning the graph is free of odd antiholes of length at least seven. Next we treat odd holes of length five that are contained in the neighborhood of a vertex. The resulting graph is quasi-line. Finally, we solve the problem using a recent characterization of quasi-line graphs by Chudnovsky and Seymour [3] and related algorithmic results of King and Reed [13]. In Section 5 we mention relevant open problems.

2. Preliminaries

All graphs in this paper are undirected, finite, and neither have loops nor multiple edges. Let G be a graph. We refer to the vertex set and edge set of G by $V = V(G)$ and $E = E(G)$, respectively. The *neighborhood* of a vertex u in G is denoted by $N_G(u) = \{v \in V \mid uv \in E\}$. The subgraph of G induced by $U \subseteq V$ is denoted $G[U]$. Analogously, the *neighborhood* of a set $U \subseteq V$ is $N(U) := \bigcup_{u \in U} N(u) \setminus U$. We say that two vertex-disjoint subsets of V are *adjacent* if some of their vertices are adjacent. The *distance* $d(u, v)$ between two vertices u and v in G is the number of edges on a shortest path between them. The *edge contraction* of an edge $e = uv$ removes its two end vertices u, v and replaces it by a new vertex adjacent to all vertices in $N(u) \cup N(v)$ (without introducing loops or multiple edges).

We denote the path and cycle on n vertices by P_n and C_n, respectively. Let $P = v_1v_2 \dots v_p$ be a path with a fixed orientation. The successor v_{i+1} of v_i is denoted by v_i^+ and its predecessor v_{i-1} by v_i^-. The segment $v_iv_{i+1} \dots v_j$ is denoted by $v_i\overrightarrow{P}v_j$. The converse segment $v_jv_{j-1} \dots v_i$ is denoted by $v_j\overleftarrow{P}v_i$.

A *hole* is an induced cycle of length at least 4 and an *antihole* is the complement of a hole. We say that a hole is *odd* if it has an odd number of edges. An antihole is called odd if it is the complement is an odd hole.

A *claw* is the graph $(\{x, a, b, c\}, \{xa, xb, xc\})$, where vertex x is called the *center* of the claw. A graph is *claw-free* if it does not contain a claw as an induced subgraph. A *clique* is a subgraph isomorphic to a complete graph. A *diamond* is a graph obtain from a clique on four vertices after removing one edge. A vertex u in a graph G is *simplicial* if $G[N(u)]$ is a clique.

Let s and t be two specified vertices in a graph $G = (V, E)$. A vertex $v \in V$ is called *irrelevant* for vertices s and t if v does not lie on any induced path from s to t. A graph G is *clean* if none of its vertices is irrelevant. We say that we *clean* G for s and t by repeatedly deleting irrelevant vertices for s and t as long as possible. In general, determining if a vertex is irrelevant is NP-complete [1]. However, for claw-free graphs, the authors of [12] could show the following (where they used Observation 1.2 and Theorem 2.7 for obtaining the polynomial time bound).

Lemma 2.1 ([12])**.** *Let s, t be two vertices of a claw-free graph G. Then G can be cleaned for s and t in polynomial time. Moreover, the resulting graph does not contain an odd antihole of length at least seven.*

The *line graph* of a graph G with edges $e_1, \dots, e_p$ is the graph $L = L(G)$ with vertices $u_1, \dots, u_p$ such that there is an edge between any two vertices u_i and u_j if and only if e_i and e_j share an end vertex in H. We note that mutually induced paths in a line graph $L(G)$ are in one-to-one correspondence with vertex-disjoint paths in G. Combining this observation with Theorem 1.3 leads to the following result.

Corollary 2.2. *For fixed k, the k-*Induced Disjoint Paths *problem can be solved in polynomial time in line graphs.*

A graph $G = (V, E)$ is called a *quasi-line graph* if for every vertex $u \in V$ there exist two vertex-disjoint cliques A and B in G such that $N(u) = V(A) \cup V(B)$ (where $V(A)$ and $V(B)$ might be adjacent). Clearly, every line graph is quasi-line and every quasi-line graph is claw-free. The following observation is useful and easy to see by looking at the complements of neighborhood in a graph.

Observation 2.3. A claw-free graph G is a quasi-line graph if and only if G does not contain a vertex with an odd antihole in its neighborhood.

A clique in a graph G is called *nontrivial* if it contains at least two vertices. A nontrivial clique A is called *homogeneous* if every vertex in $V(G)\backslash V(A)$ is either adjacent to all vertices of A or to none of them. Notice that it is possible to check in polynomial time if an edge of the graph is a homogeneous clique. This justifies the following observation.

Observation 2.4. The problem of detecting a homogeneous clique in a graph is solvable in polynomial time.

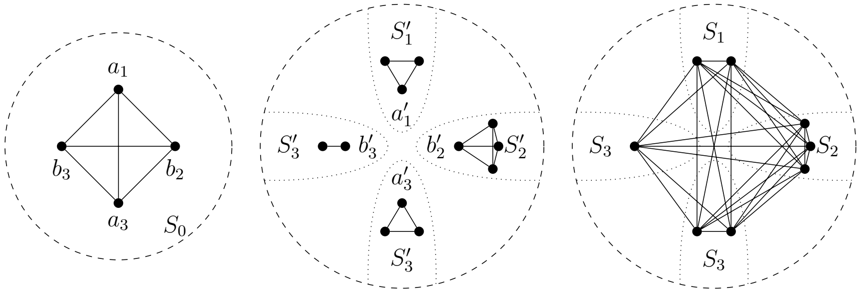


Figure 1: Composition of three linear interval strips (only part of the graph is displayed).

Two disjoint cliques A and B form a *homogeneous pair* in G if the following two conditions hold. First, at least one of A, B contains more than one vertex. Second, every vertex $v \in V(G) \setminus (V(A) \cup V(B))$ is either adjacent to all vertices of A or to none vertex of A as well as either adjacent to all of B or to none of B. The following result by King and Reed [13, Section 3] will be useful.

Lemma 2.5 ([13])**.** *The problem of detecting a homogeneous pair of cliques in a graph is solvable in polynomial time.*

Let V be a finite set of points of a real line, and $\mathcal{I}$ be a collection of intervals. Two points are adjacent if and only if they belong to a common interval $I \in \mathcal{I}$. The resulting graph is a *linear interval graph*. Analogously, if we consider a set of points of a circle and set of intervals (angles) on the circle we get a *circular interval graph*. Graphs in both classes are claw-free, in fact linear interval graphs coincide with proper interval graphs (intersection graph of a set of intervals on a line, where no interval contains another from the set) and circular interval graphs coincide with proper circular arc graphs (defined analogously). We need the following result of Deng, Hell, and Huang [5].

Theorem 2.6 ([5])**.** *Circular interval graphs and linear interval graphs can be recognized in linear time. Furthermore, a corresponding representation of such graphs can be constructed in linear time as well.*

A *linear interval strip* (S, a, b) is a linear interval graph S where a and b are the leftmost and the rightmost points (vertices) of its representation. Observe that in such a graph the vertices a and b are simplicial. Let S_0 be a graph with vertices $a_1, b_1, \ldots, a_n, b_n$ that is isomorphic to an arbitrary disjoint union of complete graphs. Let $(S'_1, a'_1, b'_1), \ldots, (S'_n, a'_n, b'_n)$ be a collection of linear interval strips. The *composition* S_n is defined inductively where S_i is formed from the disjoint union of S_{i-1} and S'_i, where:

- all neighbors of a_i are connected to all neighbors of a'_i;
- all neighbors of b_i are connected to all neighbors of b'_i;
- vertices a_i, a'_i, b_i, b'_i are removed.

See Figure 1 for an example. We are now ready to state the structure of quasi-line graphs as characterized by Chudnovsky and Seymour [3].

Theorem 2.7 ([3])**.** *A quasi-line graph G with no homogeneous pair of cliques is either a circular interval graph or a composition of linear interval strips.*

Finally, we need another algorithmic result of King and Reed [13]. They observe that the composition of the final strip in a composition of linear interval graphs is a so-called nontrivial interval 2-join and that every nontrivial interval 2-join contains a so-called canonical interval 2-join. In Lemma 13 of this paper they show how to find in polynomial time a canonical interval 2-join in a quasi-line graph with no homogeneous pair of cliques and no simplicial vertex or else to conclude that none exists. Recursively applying this result leads to the following lemma.

Lemma 2.8 ([13]). *Let G be a quasi-line graph with no homogeneous pairs of cliques and no simplicial vertex that is a composition of linear interval strips. Then the collection of linear interval strips that define G can be found in polynomial time.*

3. Our Main Result

Here is our main result.

Theorem 3.1. *For any fixed k, the k-in-a-Path problem is solvable in polynomial time in claw-free graphs.*

In order to prove Theorem 3.1 we define the following problem.

Ordered-k-in-a-Path
Instance: a graph G with k terminals ordered as $t_1, \dots, t_k$.
Question: does there exist an induced path of G starting in t_1 then passing through $t_2, \dots, t_{k-1}$ and ending in t_k?

We can resolve the original k-in-a-Path problem by $k!$ rounds of the more specific version defined above, where in each round we order the terminals by a different permutation. Hence, since we assume that k is fixed, it suffices to prove Theorem 3.2 in order to obtain Theorem 3.1.

Theorem 3.2. *For any fixed k, the* Ordered-k-in-a-Paths *problem is solvable in polynomial time in claw-free graphs.*

We prove Theorem 3.2 in Section 4 and finish this section with the following consequence of it.

Corollary 3.3. *For any fixed k, the k-*Disjoint Induced Paths *and k-*Induced Cycle *problem are solvable in polynomial time in claw-free graphs.*

Proof. Let G be a claw-free graph that together with terminals $t_1, \dots, t_k$ is an instance of k-Induced Cycle. We fix an order of the terminals, say, the order is $t_1, \dots, t_k$. We fix neighbors a_i and b_{i-1} of each terminal t_i. This way we obtain an instance of k-Induced Disjoint Paths with pairs of terminals (a_i, b_i) where $b_0 = b_k$. Clearly, the total number of instances we have created is polynomial. Hence, we can solve k-Induced Cycle in polynomial time if we can solve k-Induced Disjoint Paths in polynomial time.

Let G be a claw-free graph that together with k pairs of terminals (a_i, b_i) for $i = 1, \dots, k$ is an instance of the k-Induced Disjoint Paths problem. First we add an edge between each pair of non-adjacent neighbors of every terminal in $T = \{a_1, \dots, a_k, b_1, \dots, b_k\}$. We denote the resulting graphs obtained after performing this operation on a terminal by $G_1, \dots, G_{2k}$, and define $G_0 := G$. We claim that $G' = G_{2k}$ is claw-free and prove this by induction.

The claim is true for G_0. Suppose the claim is true for G_j for some $0 \leq j \leq 2k - 1$. Consider G_{j+1} and suppose, for contradiction, that G_{j+1} contains an induced subgraph isomorphic to a claw. Let $K := \{x, a, b, c\}$ be a set of vertices of G_{j+1} inducing a claw with center x. Let $s \in T$ be the vertex of G_j that becomes simplicial in G_{j+1}. Then $x \neq s$. Since G_j is claw-free, we may without loss of generality assume that at least two vertices of K must be in $N_{G_{j+1}}(s) \cup \{s\}$. Since $N_{G_{j+1}}(s) \cup \{s\}$ is a clique of G_{j+1} and $\{a, b, c\}$ is an independent set of G_{j+1}, we may without loss of generality assume that $K \cap (N_{G_{j+1}}(s) \cup \{s\}) = \{x, a\}$ and $\{b, c\} \subseteq V(G_{j+1}) \setminus (N_{G_{j+1}}(s) \cup \{s\})$. Then $\{x, b, c, s\}$ induces a claw in G_j with center x, a contradiction. Hence, G' is indeed claw-free.

We note that G with terminals $(a_1, b_1), \dots, (a_k, b_k)$ forms a Yes-instance of k-Induced Disjoint Paths if and only if G' with the same terminal pairs is a Yes-instance of this problem. In the next step we identify terminal b_i with a_{i+1}, i.e., for $i = 1, \dots, k-1$ we remove b_i, a_{i+1} and replace them by a new vertex t_{i+1} adjacent to all neighbors of a_{i+1} and to all neighbors of b_i. We call the resulting graph G'' and observe that G is claw-free. We define $t_1 := a_1$ and $t_{k+1} := b_k$ and claim that G' with terminal pairs $(a_1, b_1), \dots, (a_k, b_k)$ forms a Yes-instance of the k-Induced Paths problem if and only if G'' with terminals $t_1, \dots, t_{k+1}$ forms a Yes-instance of the Ordered-$(k+1)$-in-a-Path problem.

In order to see this, suppose G' contains k mutually induced paths P_i such that P_i connects a_i to b_i for $1 \leq i \leq k$. Then

$$P = t_1 \overrightarrow{P_1} b_1^- t_2 a_2^+ \overrightarrow{P_2} b_2^- \dots t_k a_k^+ \overrightarrow{P_k} t_k$$

is an induced path passing through the terminals t_i in prescribed order. Now suppose G'' contains an induced path P passing through terminals in order $t_1, \dots, t_{k+1}$. For $i = 1, \dots, k+1$ we define paths $P_i = a_i t_i^+ \overrightarrow{P} t_{i+1}^- b_i$, which are mutually induced. We now apply Theorem 3.2. This completes the proof. ■

4. The Proof of Theorem 3.2

We present a polynomial-time algorithm that solves the Ordered-k-in-a-Path problem on a claw-free graph G with terminals in order $t_1, \dots, t_k$ for any fixed integer k. We call an induced path P from t_1 to t_k that contains the other terminals in order $t_2, \dots, t_{k-1}$ a *solution* of this problem. Furthermore, an operation in this algorithm on input graph G with terminals $t_1, \dots, t_k$ *preserves the solution* if the following holds: the resulting graph G' with resulting terminals $t'_1, \dots, t'_{k'}$ for some $k' \leq k$ is a Yes-instance of the Ordered-k'-in-a-Path problem if and only if G is a Yes-instance of the Ordered-k-in-a-Path problem. We call G *simple* if the following three conditions hold:

(i) t_1, t_k are of degree one in G and all other terminals t_i $(1 < i < k)$ are of degree two in G, and the two neighbors of such t_i are not adjacent;

(ii) the distance between any pair t_i, t_j is at least four;

(iii) G is connected.

The Algorithm and Proof of Theorem 3.2

Let G be an input graph with terminals $t_1, \dots, t_k$.

If $k = 2$, we compute a shortest path from t_1 to t_2. If $k = 3$, we use Theorem 1.1 together with Observation 1.2. Suppose $k \geq 4$.

Step 1. Reduce to a set of simple graphs.
We apply Lemma 4.1 and obtain in polynomial time a set $\mathcal{G}$ that consists of a polynomial

number of simple graphs of size at most $|V(G)|$ such that there is a solution for G if and only if there is a solution for one of the graphs in $\mathcal{G}$. We consider *each* graph in $\mathcal{G}$. For convenience we denote such a graph by G as well.

Step 2. Reduce to a quasi-line graph.
We first clean G for t_1 and t_k. If during cleaning we remove a terminal, then we output No. Otherwise, clearly, we preserve the solution. By Lemma 2.1, this can be done in polynomial time and ensures that there are no odd antiholes of length at least seven left. Also, G stays simple. Then we apply Lemma 4.2, which removes vertices v whose neighborhood contain an odd hole of length five, as long as we can. Clearly, we can do this in polynomial time. Note that G stays connected since we do not remove cut-vertices due to the claw-freeness. By condition (i), we do not remove a terminal either. Afterwards, we clean G again for t_1 and t_k. If we remove a terminal, we output No. Otherwise, as a result of our operations, G becomes a simple quasi-line graph due to Observation 2.3.

Step 3. Reduce to a simple quasi-line graph with no homogeneous clique
We first exhaustively search for homogeneous cliques by running the polynomial algorithm mentioned in Observation 2.4 and apply Lemma 4.3 each time we find such a clique. Clearly, we can perform the latter in polynomial time as well. After every reduction of such a clique to a single vertex, G stays simple and quasi-line, and at some moment does not contain any homogeneous clique anymore, while we preserve the solution.

Step 4. Reduce to a circular interval graph or to a composition of interval strips.
Let t'_1, t'_k be the (unique) neighbor of t_1 and t'_k, respectively. As long as G contains homogeneous pairs of cliques (A, B) so that A neither B is equal to $\{t_1, t'_1\}$ or $\{t_k, t'_k\}$, we do as follows. We first detect such a pair in polynomial time using Lemma 2.5 and reduce them to a pair of single vertices by applying Lemma 4.4. Also performing Lemma 4.4 clearly takes only polynomial time. After every reduction, G stays simple and quasi-line, and we preserve the solution. At some moment, the only homogeneous pairs of cliques that are possibly left in G are of the form $(\{t_1, t'_1\}, B)$ and $(\{t_k, t'_k\}, B)$. As G does not contain a homogeneous clique (see Step 3), the cliques in such pairs must have adjacent vertex sets. Hence, there can be at most two of such pairs. We perform Lemma 4.4 and afterwards make the graph simple again. Although this might result in a number of new instances, their total number is still polynomial because we perform this operation at most twice. Hence, we may without loss of generality assume that G stays simple. By Theorem 2.7, G is either a circular interval graph or a composition of linear interval strips; we deal with theses two cases separately after recognizing in polynomial time in which case we are by using Theorem 2.6.

Step 5a. Solve the problem for a circular interval graph.
Let G be a circular interval graph. Observe that the order of vertices in an induced path must respect the natural order of points on a circle. Hence, deleting all points that lie on the circle between t_k and t_1 preserves the solution. So, we may even assume that G is a linear interval graph. We solve the problem in these graphs in Theorem 4.5.

Step 5b. Solve the problem for a composition of linear interval strips.
Let G be a composition of linear interval strips. Because G is assumed to be clean for $t_1, \ldots, t_k$, G contains no simplicial vertex. Then we can find these strips in polynomial time using Lemma 2.8 and use this information in Lemma 4.6. There we create a line

graph G' with $|V(G')| \leq |V(G)|$, while preserving the solution. Moreover, this can be done in polynomial time by the same theorem. Then we use Corollary 2.2 to prove that the problem is polynomially solvable in line graphs in Theorem 4.7.

Now it remains to state and prove Lemmas 4.1–4.6 and Theorems 4.5– 4.7.

Lemma 4.1. *Let G be a graph with terminals ordered $t_1, \ldots, t_k$. Then there exists a set $\mathcal{G}$ of $n^{O(k)}$ simple graphs, each of size at most $|V(G)|$, such that G has a solution if and only if there exists a graph in $\mathcal{G}$ that has a solution. Moreover, $\mathcal{G}$ can be constructed in polynomial time.*

Proof. We branch as follows. First we guess the first six vertices after t_1 in a possible solution. Then we guess the last six vertices before t_n. Finally, for $2 \leq i \leq n-1$, we guess the last six vertices preceding t_i and the first six vertices following t_i. We check if the subgraph induced by the terminals and all guessed vertices has maximum degree 2. If not we discard this guess. Otherwise, for every terminal and for every guessed vertex that is not an end vertex of a guessed subpath, we remove all its neighbors that are not guessed vertices. This way we obtain a number of graphs which we further process one by one.

Let G' be such a created subgraph. If G' does not contain all terminals, we discard G'. If G' is disconnected then we discard G' if two terminals are in different components, or else we continue with the component of G' that contains all the terminals. Suppose there is a guessed subpath in G' containing more than one terminal. If the order is not $t_i, t_{i+1}, \ldots, t_j$ for some $i < j$, we discard G'. Otherwise, if necessary, we place t_i and t_j on this subpath such that they are at distance at least four of each other and also are of distance at least four of each end vertex of the subpath. Because the guessed subpaths are sufficiently long, such a placement is possible. We then remove $t_{i+1}, \ldots, t_{j-1}$ from the list of terminals. After processing all created graphs as above, we obtain the desired set $\mathcal{G}$. Since k is fixed, $\mathcal{G}$ can be constructed in polynomial time. ∎

Lemma 4.2. *Let G be a simple claw-free graph. Removing a vertex $u \in V(G)$, the neighborhood of which contains an induced odd hole of length five, preserves the solution.*

Proof. Because G is simple, u is not a terminal. We first show the following claim.

Claim 1. Let $G[\{v, w, x, y\}]$ be a diamond in which vw is a non-edge. If there is a solution P that contains v, x, w, then there is another solution that contains v, y, w (and that does not contain x).

In order to see this take the original solution P and notice that by claw-freeness any neighbor of y on P must be in the (closed) neighborhood of v or w. This way the solution can be rerouted via y, without using x. This proves Claim 1.

Now suppose that u is a vertex which has an odd hole C of length five in its neighborhood. Obviously, G is a YES-instance if $G - u$ is a YES-instance. To prove the reverse implication, suppose G is a YES-instance. Let P be a solution. If u does not belong to P then we are done. Hence, we suppose that u belongs to P and consider three cases depending on $|V(C) \cap V(P)|$.

Case 1. $|V(C) \cap V(P)| \geq 2$. Then $|V(C) \cap V(P)| = 2$, as any vertex on P will have at most two neighbors. We are done by Claim 1.

Case 2. $|V(C) \cap V(P)| = 1$. Let $w \in V(C)$ belong to P and let the other neighbor of u that belongs to P be x. We note that x must be adjacent to at least one of the neighbors of w in C. Then we can apply Claim 1 again.

Case 3. $|V(C) \cap V(P)| = 0$. Let the two neighbors of u on P be x and y. To avoid a claw at u, every vertex of C must be adjacent to x or y. If there is a vertex in C adjacent to both, we apply Claim 1. Suppose there is no such vertex and that the vertices of the C are partitioned in two sets X (vertices of C only adjacent to x) and Y (vertices of C only adjacent to y). We assume without loss of generality that $|X| = 3$, and hence contains a pair of independent vertices which together with u and y form a claw. This is a contradiction. ■

Lemma 4.3. *Let G be a simple quasi-line graph with a homogeneous clique A. Then contracting A to a single vertex preserves the solution and the resulting graph is a simple quasi-line graph containing the same terminals as G.*

Proof. Each vertex in A lies on a triangle, unless G is isomorphic to P_2, which is not possible. Hence, by condition (i), A does not contain a terminal. We remove all vertices of A except one. The resulting graph will be a simple quasi-line graph containing the same terminals, and we will preserve the solution. ■

Lemma 4.4. *Let G be a simple quasi-line graph with terminals ordered $t_1, \ldots, t_k$ that has no homogeneous clique. Contracting the cliques A and B in a homogeneous pair to single vertices preserves the solution. The resulting graph is quasi-line; it is simple unless A or B consists of two vertices u, u' with $u \in \{t_1, t_k\}$ and $d(u', t_i) \leq 3$ for some $t_i \neq u$.*

Proof. Because G does not contain a homogeneous clique, $V(A)$ and $V(B)$ must be adjacent. Then, due to condition (ii), there can be at most one terminal in $V(A) \cup V(B)$. In all the cases discussed below we will actually not contract edges but only remove vertices from A and B. Hence, the resulting graph will always be a quasi-line graph.

Suppose A contains t_1 or t_k, say t_1. Suppose $|V(A)| = 1$, so A only contains t_1. Then the neighbor of t_1 is in B and $|V(B)| \geq 2$. We delete all vertices from B except this neighbor, because they will not be used in any solution. Clearly, the resulting graph is simple and the solution is preserved. Suppose $|V(A)| \geq 2$. Because t_1 is of degree one, A consists of two vertices, namely t_1 and its neighbor t'_1. Note that t'_1 does not have a neighbor outside A and B, as t_1 is of degree one. As $V(A)$ and $V(B)$ are adjacent, t'_1 has a neighbor u in B. We delete t_1 and replace it by t'_1 in the set of terminals. We delete all vertices of B except u, because of the following reasons. If these vertices are not adjacent to t'_1, they will never appear in any solution. If they are adjacent to t'_1, they will not appear in any solution together with u, and as such they can be replaced by u. Note that t'_1 has degree one in the new graph and that this graph is only simple if $d(t'_1, t_j) \geq 4$ for all $2 \leq j \leq k$. Clearly, the solution is preserved.

Suppose A contains a terminal t_i for some $2 \leq i \leq k-1$. Suppose A only contains t_i. Because $V(A)$ and $V(B)$ are adjacent, t_i is adjacent to a vertex u in B. By condition (i), u is the only vertex in B adjacent to t_i. We delete all vertices of B except u. Clearly, the resulting graph is simple and the solution is preserved. Suppose $|V(A)| \geq 2$. By condition (ii), A contains only one other vertex t'_i and t_i, t'_i do not have a common neighbor. Then A must be separated of the rest of the graph by B. Furthermore, the other neighbor of t_i must be in B. We delete t'_i and all vertices of B except the neighbor of t_i. Clearly, the resulting graph is simple and the solution is preserved.

Suppose A does not contain a terminal. By symmetry, we may assume that B does not contain a terminal either. Let $a'b' \in E(G)$ with $a' \in V(A)$ and $b' \in V(B)$. Let G' be the graph obtained from G by removing all vertices of A except a' and B except a', b'. Note that we have kept all terminals and that the resulting graph is simple. Any solution P' for G' is a solution for G.

Now assume we have a solution P for G. We claim that $|P \cap A| \leq 1$ and $|P \cap B| \leq 1$. Suppose otherwise, say $|P \cap A| \geq 2$. Then $|P \cap A| = 2$, as P is a path. Since t_1 and t_k are not in A, we find that P contains a subpath $xuvy$ with $u, v \in A$. Since x is adjacent to $u \in A$, but also non-adjacent to $v \in A$, we find that $x \in B$. Analogously we get that $y \in B$. However, then $xy \in E(G)$. This is a contradiction.

Suppose $|P \cap A| = 0$ and $|P \cap B| = 0$. Then P is a solution for G' as well. Suppose $|P \cap A| = 0$ and $|P \cap B| = 1$. Then we may without loss of generality assume that $b' \in V(P)$ and find that P is a solution for G' as well. The case $|P \cap A| = 1$ and $|P \cap B| = 0$ follows by symmetry. Suppose $|P \cap A| = |P \cap B| = 1$, say P intersects A in a and B in b. If $ab \in E(G)$ then we replace ab by $a'b'$ and obtain a solution for G'. Suppose $ab \notin E(G)$. Because a is not a terminal, a has neighbors x and y on P. If $x, y \notin N(b)$ then $\{a', x, y, b'\}$ induces a claw in G with center a'. This is not possible. Hence, we may assume without loss of generality that y is adjacent to b. Since A or B contains at least two vertices, y has degree at least three. Then y is not a terminal. Thus we can skip y and exchange ayb in P with $a'b'$ to get the desired induced path P'. ■

Theorem 4.5. *The* Ordered-k-in-a-Path *problem can be solved in polynomial time in linear interval graphs.*

Proof. Let G be a linear interval graph. We may assume without loss of generality that the terminals form an independent set. We use its linear representation that we obtain in polynomial time by Lemma 2.8. In what follows the notions of predecessors (left) and successors (right) are considered for the linear ordering of the points on the line. Without loss of generality we may assume that t_1 is the first point and that t_k is the last and that no two points coincide. By our assumption, t_i and t_{i+1} are nonadjacent. From the set of points belonging to the closed interval $[t_i, t_{i+1}]$ we remove all neighbors of t_i except the rightmost one and all neighbors of t_{i+1} except the leftmost. Then the shortest path between t_i and t_{i+1} is induced. In addition, these partial paths combined together provide a solution unless for some terminal t_i its leftmost predecessor is adjacent to its rightmost successor. Hence, no induced path may have t_i among its inner vertices. ■

Lemma 4.6 (proof postponed to journal version)**.** *Let G be a composition of linear interval strips. It is possible to create in polynomial time a line graph G' with $|V(G')| \leq |V(G)|$, while preserving the solution.*

Theorem 4.7. *For fixed k,* Ordered-k-in-a-Path *is polynomially solvable in line graphs.*

Proof. A version of Ordered-k-in-a-Path in which the path is not necessarily induced can be easily translated into an instance of the k-Disjoint Paths problem and solved in polynomial time due to Theorem 1.3. Noting that mutually induced paths in a line graph $L(G)$ are in one-to-one correspondence with vertex-disjoint paths in G enables us to solve the Ordered-k-in-a-Path problem in polynomial time for line graphs. ■

5. Conclusions and Further Research

We showed that, for any fixed k, the problems k-in-a-Path, k-Disjoint Induced Paths and k-Induced Cycle are polynomially solvable on claw-free graphs. If k is part of the input these problems are known to be NP-complete. In the journal version we show this is true, even when the input is restricted to be claw-free. Perhaps the two most fascinating related open problems are to determine the complexity of deciding if a graph contains an odd hole (whereas the problem of finding an even hole is polynomially solvable [2]) and to determine the computational complexity of deciding if a graph contains two mutually induced holes (whereas it is known that the case of two mutually induced odd holes is NP-complete [10]). For claw-free graphs these two problems are solved. Shrem et al. [19] even obtained a polynomial-time algorithm for detecting a shortest odd hole in a claw-free graph. In the journal version we will address the second problem for claw-free graphs.

References

[1] D. Bienstock. On the complexity of testing for odd holes and induced odd paths. *Discrete Mathematics* **90** (1991) 85–92, See also Corrigendum, Discrete Mathematics **102** (1992) 109.

[2] M. Chudnovsky, K. Kawarabayashi and P.D. Seymour. Detecting even holes. *Journal of Graph Theory* **48** (2005) 85–111.

[3] M. Chudnovsky and P.D. Seymour. The structure of claw-free graphs. In *Surveys in combinatorics 2005*, Cambridge (2005) 153–171.

[4] M. Chudnovsky and P.D. Seymour. The three-in-a-tree problem. *Combinatorica*, to appear.

[5] X. Deng, P. Hell, and J. Huang. Linear time representation algorithm for proper circular-arc graphs and proper interval graphs. *SIAM Journal on Computing* **25** (1996) 390–403.

[6] N. Derhy and C. Picouleau. Finding induced trees. *Discrete Applied Mathematics* **157** (2009) 3552–3557.

[7] N. Dehry, C. Picouleau, and N. Trotignon. The four-in-a-tree problem in triangle-free graphs. *Graphs and Combinatorics* **25** (2009) 489–502.

[8] R. Faudree, E. Flandrin, and Z. Ryjáček. Claw-free graphs—a survey. *Discrete Mathematics* **164** (1997) 87–147.

[9] M.R. Fellows. The RobertsonSeymour theorems: A survey of applications. In: *Proceedings of AMS-IMS-SIAM Joint Summer Research Conf. Contemporary Mathematics*, Providence, RI (1989) 1-18.

[10] P. Golovach, M. Kamiński, D. Paulusma, and D. M. Thilikos. Induced packing of odd cycles in a planar graph. In: *Proceedings of ISAAC 2009*, LNCS 5878 (2009) 514–523.

[11] R. Haas and M. Hoffmann. Chordless paths through three vertices. *Theoretical Computer Science* **351** (2006) 360–371.

[12] P. van 't Hof, M. Kamiński and D. Paulusma. Finding induced paths of given parity in claw-free graphs. In: *Proceedings of WG 2009*, LNCS, to appear.

[13] A. King and B. Reed. Bounding χ in terms of ω and δ for quasi-line graphs. *Journal of Graph Theory* **59** (2008) 215-228.

[14] Y. Kobayashi and K. Kawarabayashi. The induced disjoint paths problem. In: *Proceedings of IPCO 2008*, LNCS 5035 (2008) 47–61.

[15] Y. Kobayashi and K. Kawarabayashi. Algorithms for finding an induced cycle in planar graphs and bounded genus graphs. In: *Proceedings of SODA 2009* (2009) 1146–1155.

[16] B. Lévêque, D.Y. Lin, F. Maffray, and N. Trotignon. Detecting induced subgraphs. *Discrete Applied Mathematics* **157** (2009) 3540–3551.

[17] N. Robertson and P.D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B* **63** (1995) 65–110.

[18] M. Chudnovsky, N. Robertson, P.D. Seymour, and R. Thomas. The strong perfect graph theorem. *Annals of Mathematics* **164** (2006) 51–229.

[19] S. Shrem, M. Stern and M.C. Golumbic. Smallest odd holes in claw-free graphs. In *Proceedings of WG 2009*, LNCS 5911 (2009) 329–340.

[20] N. Trotignon and L. Wei. The k-in-a-tree problem for graphs of girth at least k, manuscript.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 383-394
www.stacs-conf.org

FINDING INDUCED SUBGRAPHS VIA MINIMAL TRIANGULATIONS

FEDOR V. FOMIN[1] AND YNGVE VILLANGER[1]

[1] Department of Informatics,
University of Bergen, Norway
E-mail address: {fedor.fomin|yngve.villanger}@ii.uib.no

Abstract. Potential maximal cliques and minimal separators are combinatorial objects which were introduced and studied in the realm of minimal triangulation problems including Minimum Fill-in and Treewidth. We discover unexpected applications of these notions to the field of moderate exponential algorithms. In particular, we show that given an n-vertex graph G together with its set of potential maximal cliques, and an integer t, it is possible in time the number of potential maximal cliques times $O(n^{O(t)})$ to find a maximum induced subgraph of treewidth t in G and for a given graph F of treewidth t, to decide if G contains an induced subgraph isomorphic to F. Combined with an improved algorithm enumerating all potential maximal cliques in time $O(1.734601^n)$, this yields that both the problems are solvable in time $1.734601^n * n^{O(t)}$.

1. Introduction

One of the most fundamental problems in Graph Algorithms is, for a given graph $G = (V, E)$, to find a maximum or minimum subset S of V that satisfies some property Π. For example, when S is required to be a maximum set of pairwise adjacent vertices this is the Maximum Clique problem. When S is required to be a maximum set of pairwise non-adjacent vertices this is the Maximum Independent Set problem. Its complement, the Minimum Vertex Cover problem, is to find a minimum set S such that the graph $G \setminus S$ is an independent set. Another examples are Maximum Induced Forest, where one is seeking for a set of vertices inducing a forest of maximum size, or its complement Minimal Feedback Vertex Set which is to remove the minimum number of vertices to destroy all cycles.

All these examples are special cases of the problem, where one seeks a maximum subset of vertices that induces a subgraph of G from some given graph class $\mathcal{C}$. If G is an n-vertex graph, and recognition of graphs from $\mathcal{C}$ can be done in polynomial time, then the trivial brute force algorithm solves the problem in time $2^n n^{\mathcal{O}(1)}$. One of the crucial questions in

1998 ACM Subject Classification: Algorithm Analysis, Combinatorics, Data Structures, Graph Algorithms, Graph Theory.

Key words and phrases: Bounded treewidth, minimal triangulation, moderately exponential time algorithms.

Partially supported by the Norwegian Research Council.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2470

the area of moderate exponential algorithms is if the brute force algorithm can be avoided to solve any hard (NP-hard, $\#P$, PSPACE-hard, etc.) problem. So far we are still very far from answering this question. For some problems we know how to avoid the brute force search, and for some problems, like SAT, it is a big open problem in the area. Similar situation is with the problem of finding a maximum induced subgraph from a given class $\mathcal{C}$. For some simple graph classes $\mathcal{C}$ the trivial 2^n-barrier has been broken. The most well studied case is when $\mathcal{C}$ is the class of graphs without edges, or the class of graphs of treewidth 0. In this case, we are looking for an independent set of maximum size. This is the classical NP-hard problem and it is well studied in the realm of moderate exponential algorithms. The classical result of Moon and Moser [19] (see also Miller and Muller [18]) from the 1960s can be easily turned into algorithms finding a maximum independent set in time $3^{n/3}n^{\mathcal{O}(1)}$. Tarjan and Trojanowski [25] gave a $O(2^{n/3})$ time algorithm. There were several non-trivial steps in improving the running time of the algorithm including the work of Jian [17], Robson [23], and Grandoni et al. [11]. A significant amount of research was also devoted to algorithms for the Maximum Independent Set problem on sparse graphs, some examples are [7, 14, 21]. It is easy to show that a simple branching algorithm can compute a maximum induced path or cycle in time $3^{n/3}n^{\mathcal{O}(1)}$. However, breaking the 2^n-barrier even for the case when the class $\mathcal{C}$ is a forest, i.e. the class of graphs of treewidth 1, was an open problem in the area until very recently. The first exact algorithm breaking the trivial 2^n-barrier is due to Razgon [20]. The running time $\mathcal{O}(1.8899^n)$ of the algorithm from [20] was improved in [9, 10] to $\mathcal{O}(1.7548^n)$. All these algorithms for Maximum Independent Set and Maximum Induced Forest are so-called branching algorithms (a variation of Davis-Putnam-style exponential-time backtracking [8]). There is also a relevant work of Gupta et al. [15] who used branching to show that for every fixed r, there are at most c^n r-regular subgraphs for some $c < 2$. For example, for Maximum Induced Matching and Maximum 2-Regular Induced Subgraph, their results yield algorithms solving these problems in time $\mathcal{O}(1.695733^n)$ and $\mathcal{O}(1.7069^n)$, respectively. However, the results of Gupta et al. strongly depend on the regularity of the maximum subgraphs. To our knowledge, prior to our work no algorithms better than the trivial brute-force $\mathcal{O}(2^n)$ were known for more complicated classes $\mathcal{C}$.

In this work we make a step aside the "branching" path and use a completely different approach for problems on finding induced subgraphs. Our approach is based on a tools from the area of minimal triangulations, namely, potential maximal cliques. Minimal triangulations are the result of adding an inclusion minimal set of edges to produce a triangulation (or chordal graph). The study of minimal triangulations dates back to the 1970s and originated from research on sparse matrices and vertex elimination in graphs. Minimal separators are one of the main tools in the study of minimal triangulations. We refer to the survey of Heggernes [16] for more information on triangulations. Potential maximal cliques were defined by Bouchitté and Todinca [5, 6] and were used in different algorithms for computing the treewidth of a graph [12, 13]. A subset of vertices C of a graph G is a *potential maximal clique* if there is a minimal triangulation TG of G such that C is a maximal clique in TG. At first glance it is not clear, what is the relation between potential maximal cliques and induced subgraphs. Our first main result establishes such a relation.

- Let Π_G be the set of potential maximal cliques in G. A maximum induced subgraph of treewidth t in an n-vertex graph G can be found in time $\mathcal{O}(|\Pi_G| \cdot n^{\mathcal{O}(t)})$ (Section 3).

As we already mentioned, the well studied Maximum Independent Set (and its dual Minimum Vertex Cover) and Maximum Induced Forest (and Minimum Feedback Vertex Set) are the special cases for $t = 0$ and $t = 1$, respectively. Our second main result shows that

- All potential maximal cliques can be enumerated in time $\mathcal{O}(1.734601^n)$ (Section 5).

Combining both results, we obtain that a maximum induced subgraph of treewidth t in an n-vertex graph G can be found in time $\mathcal{O}(1.734601^n \cdot n^{\mathcal{O}(t)})$. While for $t = 0$ (the case of Maximum Independent Set) the existing branching algorithms are much faster than $\mathcal{O}(1.734601^n)$, already for $t = 1$ (the case of Maximum Induced Forest) our algorithm is already faster than the best known branching algorithm [10]. For fixed $t \geq 2$, no algorithm better than the trivial $\mathcal{O}(2^n n^{\mathcal{O}(1)})$ brute force algorithm was known.

With small modifications, our algorithm can be used for other problems involving induced subgraphs. As an example, we show how to solve the induced subgraph isomorphism problem, which is to decide if G contains an induced subgraph isomorphic to a given graph F (Section 4). We show that when the treewidth of F is at most t, then this problem is solvable in time $1.734601^n \cdot n^{\mathcal{O}(t)}$. In particular, when the treewidth of F is $o(n/\log n)$, for example when F is a planar graph, or a graph excluding some fixed graph as a minor, the running time of our algorithm is $1.734601^{n+o(n)}$. Let us note that no algorithm faster than the trivial brute-force algorithm was known even when F is a tree.

Finally, our new algorithm enumerating potential maximal cliques is not only (slightly) faster than the algorithm from [13] and thus by [12], directly implies faster exact algorithm computing the treewidth of a graph. It is also significantly simpler than the previous algorithms and is easy to implement. Due to space limitations, some proofs are omitted. A full version will appear at some later point.

2. Preliminaries

We denote by $G = (V, E)$ a finite, undirected, and simple graph with $|V| = n$ vertices and $|E| = m$ edges. For any nonempty subset $W \subseteq V$, the subgraph of G induced by W is denoted by $G[W]$. For $S \subseteq V$ we often use $G \setminus S$ to denote $G[V \setminus S]$. The *neighborhood* of a vertex v is $N(v) = \{u \in V : \{u, v\} \in E\}$, $N[v] = N(v) \cup \{v\}$, and for a vertex set $S \subseteq V$ we set $N(S) = \bigcup_{v \in S} N(v) \setminus S$, $N[S] = N(S) \cup S$. A *clique* C of a graph G is a subset of V such that all the vertices of C are pairwise adjacent. By $\omega(G)$ we denote the maximum clique-size of a graph G.

A graph H is *chordal* (or *triangulated*) if every cycle of length at least four has a chord, i.e., an edge between two nonconsecutive vertices of the cycle. A *triangulation* of a graph $G = (V, E)$ is a chordal graph $H = (V, E')$ such that $E \subseteq E'$. Graph H is a *minimal triangulation* of G if for every edge set E'' with $E \subseteq E'' \subset E'$, the graph $F = (V, E'')$ is not chordal.

The notion of treewidth is due to Robertson and Seymour [22]. A *tree decomposition* of a graph $G = (V, E)$, denoted by $TD(G)$, is a pair (X, T) in which $T = (V_T, E_T)$ is a tree and $X = \{X_i \mid i \in V_T\}$ is a family of subsets of V, called *bags*, such that

(i) $\bigcup_{i \in V_T} X_i = V$;

(ii) for each edge $e = \{u, v\} \in E$ there exists an $i \in V_T$ such that both u and v belong to X_i;

(iii) for all $v \in V$, the set of nodes $\{i \in V_T \mid v \in X_i\}$ induces a connected subtree of T.

The maximum of $|X_i|-1$, $i \in V_T$, is called the *width* of the tree decomposition. The *treewidth* of a graph G, denoted by $\mathbf{tw}(G)$, is the minimum width taken over all tree decompositions of G.

Theorem 2.1 (folklore)**.** *For any graph G, $\mathbf{tw}(G) \leq k$ if and only if there is a triangulation H of G such that $\omega(H) \leq k+1$.*

Let u and v be two non adjacent vertices of a graph $G=(V,E)$. A set of vertices $S \subseteq V$ is a *u,v-separator* if u and v are in different connected components of the graph $G[V \setminus S]$. A connected component C of $G[V \setminus S]$ is a *full* component associated to S if $N(C)=S$. Separator S is a *minimal u,v-separator* of G if no proper subset of S is a u,v-separator. Notice that a minimal separator can be strictly included in another one. We denote by Δ_G the set of all minimal separators of G.

A set of vertices $\Omega \subseteq V$ of a graph G is called a *potential maximal clique* if there is a minimal triangulation H of G such that Ω is a maximal clique of H. We denote by Π_G the set of all potential maximal cliques of G.

For a minimal separator S and a full connected component C of $G \setminus S$, we say that (S,C) is a *block* associated to S. We sometimes use the notation (S,C) to denote the set of vertices $S \cup C$ of the block. It is easy to see that if $X \subseteq V$ corresponds to the set of vertices of a block, then this block (S,C) is unique: indeed, $S=N(V \setminus X)$ and $C = X \setminus S$.

We also need the following result of Bouchitté and Todinca on the structure of potential maximal cliques.

Theorem 2.2 (Bouchitté and Todinca [5])**.** *Let $K \subseteq V$ be a set of vertices of the graph $G=(V,E)$. Let $\mathcal{C}(K)=\{C_1,\dots,C_p\}$ be the set of connected components of $G \setminus K$ and let $\mathcal{S}(K)=\{S_1,S_2,\dots,S_p\}$, where $S_i=N(C_i)$, $i \in \{1,2,\dots,p\}$, is the set of those vertices of K which are adjacent to at least one vertex of the component C_i. Then K is a potential maximal clique of G if and only if*

1. *$G \setminus K$ has no full component associated to K, and*
2. *the graph on the vertex set K obtained from $G[K]$ by completing each $S_i \in \mathcal{S}(K)$ into a clique is a complete graph.*

Moreover, if K is a potential maximal clique, then $\mathcal{S}(K)$ is the set of minimal separators of G contained in K.

3. Induced subgraph of bounded treewidth

In this section we prove the first result relating the problems of finding an induced subgraph and enumerating potential maximal cliques. The following lemma is crucial for our algorithm.

Lemma 3.1. *Let $F=(V_F,E_F)$ be an induced subgraph of a graph $G=(V_G,E_G)$. Then for every minimal triangulation TF of F, there is a minimal triangulation TG of G such that for every clique K of TG, the intersection $K \cap V_F$ is either empty, or is a clique of TF.*

Now we are ready to proceed with the main result of this section.

Theorem 3.2. *Let G be a graph on n vertices and m edges given together with the set Π_G of its potential maximal cliques and the set Δ_G of its minimal separators. For any integers $0 \leq t, \ell \leq n$, there is an algorithm that checks in time $\mathcal{O}(n^{t+4}m(|\Pi_G|+|\Delta_G|))$ if G contains an ℓ-vertex induced subgraph of treewidth at most t.*

Proof. Let F be an induced subgraph of treewidth at most t. By Lemma 2.1, there is a minimal triangulation TF of F, such that the size of a maximal clique of TF is at most $t+1$. By Lemma 3.1, there is a minimal triangulation TG of G, such that every clique of TG contains at most $t+1$ vertices of F. If we knew such a minimal triangulation TG, dynamic programming over the clique-tree of TG will provide the answer to our question in time $\mathcal{O}(n^{t+3}m)$. However, we are not given such a triangulation a priori. Thus, the computations require multiplicative factor $n|\Pi_G|$.

We start by enumerating all full blocks and sorting them by their sizes. This can be done by enumerating all minimal separators, and checking for each minimal separator S and each of the connected component of $G \setminus S$ if this is a full component or not. By making use of Theorem 5.6, this step can be performed in time $\mathcal{O}(|\Delta_G| \cdot n^3)$. Sorting blocks can be done in $\mathcal{O}(n|\Delta_G|)$ time using a bucket sort.

For a minimal separator S, a full block (S, C), and a potential maximal clique Ω, we call the triple (S, C, Ω) *good* if $S \subseteq \Omega \subseteq C \cup S$. For each full block we also enumerate all good triples that can be obtained from this block as follows. By Theorem 2.2, if a minimal separators S is a subset of a potential maximal clique Ω, then $S = N(C)$ for some connected component C of $G[V \setminus \Omega]$, and thus, the number of minimal separators contained in Ω is at most n. By Theorem 2.2, $G \setminus \Omega$ has no full component associated to Ω, and thus for every minimal separator $S \subseteq \Omega$, we have that $\Omega \setminus S \neq \emptyset$. Therefore, there exists a vertex $u \in \Omega \setminus S$ and thus Ω is a subset of the full block (S, C) such that $u \in C$. But this yields that every potential maximal clique is contained in at most n good triples, and the total number of good triples is at most $n|\Pi_G|$. Computing for every potential maximal clique all good triples containing it, in time $\mathcal{O}(m|\Pi_G|)$ one can create a data structure that for each full block assigns the set of potential maximal cliques that make a good triple with that block.

After preprocessing blocks and creating good triples, we proceed with dynamic programming. The dynamic programming consists of two step. In the first, most technical step, we compute the sizes of maximal subgraphs in full blocks (S, C) subject to the condition that the minimal separator S contains at most $t+1$ vertices of the subgraph. To compute these values we use deep combinatorial results of Bouchitté and Todinca on the structure of potential maximal cliques. In the second step, we go through all minimal separators, and for each separator we glue solutions found at the first step.

Step 1: Processing full blocks. We need to define several functions. For a full block (S, C), and for every subset $W \subseteq S$, $|W| \leq t+1$, and integer $0 \leq \ell \leq n$, $\alpha(\ell, W, S, C) = 1$ if there exits an induced subgraph $F = (V_F, E_F)$ of $G[C \cup W]$ such that $|V_F| = \ell$, $V_F \cap S = W$, and F has a minimal triangulation TF such that $\omega(TF) \leq t+1$ and W is a clique of TF. Otherwise, $\alpha(\ell, W, S, C) = 0$.

For every inclusion minimal block (S, C), we have that $S \cup C$ is a potential maximal clique. Thus for every inclusion minimal block (S, C), and for every set $W \subseteq S \cup C$, $|W| \leq t+1$, we put

$$\alpha(\ell, W, S, C) = \begin{cases} 1, & \text{if } \ell = |W|, \\ 0, & \text{otherwise.} \end{cases}$$

To compute the values of α for larger blocks, we perform dynamic programming over sets of good triples formed by smaller blocks. For every good triple (S, C, Ω), and for every subset $W \subseteq \Omega$, $|W| \leq t+1$, and integer $0 \leq \ell \leq n$, we want to compute an auxiliary function such that $\beta(\ell, W, S, C, \Omega) = 1$ if there exits an induced subgraphs $F = (V_F, E_F)$ of

$G[C \cup W]$ such that $|V_F| = \ell$, $V_F \cap \Omega = W$, and F has a minimal triangulation TF such that $\omega(TF) \leq t+1$, and W is a clique of TF. Otherwise, $\beta(\ell, W, S, C, \Omega) = 0$.

Let us remark that

$$\alpha(\ell, W, S, C) = 1 \Leftrightarrow \exists \text{ good triple } (S, C, \Omega) \text{ and } W \subseteq W' \subseteq \Omega \text{ s.t. } \beta(\ell, W', S, C, \Omega) = 1.$$

Indeed, if $\beta(\ell, W', S, C, \Omega) = 1$, then there is a minimal triangulation TF of an induced subgraph $F = (V_F, E_F)$ of $G[C \cup W]$ such that $|V_F| = \ell$, $\omega(TF) \leq t+1$, and W is a clique of TF, simply because this is true for W' and $W \subseteq W'$. Then $TF[V_F \setminus (W' \setminus W)]$ is the triangulation of $F[V_F \setminus (W' \setminus W)]$ that certifies $\alpha(\ell, W, S, C) = 1$. For the opposite direction the arguments are similar.

We start computing β from inclusion minimal blocks. For every inclusion minimal block (S, C), and for every set $W \subseteq S \cup C$, $|W| \leq t+1$,

$$\beta(\ell, W, S, C, \Omega) = \begin{cases} 1, & \text{if } \ell = |W|, \\ 0, & \text{otherwise.} \end{cases}$$

To compute $\beta(\ell, W, S, C, \Omega)$ we define an auxiliary function γ as follows. Let $\{C_1, \dots, C_p\}$ be the vertex sets of the connected components of $G[(S \cup C) \setminus \Omega]$. By Theorem 2.2, the sets $S_i = N(C_i)$, $1 \leq i \leq p$, are minimal separators of G, and moreover, $S_i \subset \Omega$ for $1 \leq i \leq p$. The values of function $\gamma(\ell, j, W, S, C, \Omega)$ are in $\{0, 1\}$. For every good triple (S, C, Ω), and for every subset $W \subset \Omega$, $|W| \leq t+1$, and $0 \leq \ell \leq n$, $\gamma(\ell, j, W, S, C, \Omega) = 1$ if and only if there exits an induced subgraph $F = (V_F, E_F)$ of $G[W \cup \bigcup_{i=1}^{j} C_i]$ such that $|V_F| = \ell$, $V_F \cap \Omega = W$, and F has a minimal triangulation TF such that $\omega(TF) \leq t+1$ and W is a clique in TF. Note that $G[W \cup \bigcup_{i=1}^{p} C_i] = G[W \cup C]$, and by definitions of β and γ, we have that

$$\beta(\ell, W, S, C, \Omega) = \gamma(\ell, p, W, S, C, \Omega).$$

Now for every $\ell \geq 0$,

$$\gamma(\ell, 1, W, S, C, \Omega) = \alpha(\ell - |W \setminus S_1|, W \cap S_1, S_1, C_1).$$

For $j > 1$,

$$\gamma(\ell, j, W, S, C, \Omega) = \begin{cases} 1, & \text{if } \gamma(i, j-1, W, S, C, \Omega) = 1 \wedge \alpha(\ell - i + |W \cap S_j|, \\ & \quad W \cap S_j, S_j, C_j) = 1, \text{for some } i, 1 \leq i \leq \ell, \\ 0, & \text{otherwise.} \end{cases}$$

This is because for every ℓ-vertex subgraph $F = (V_F, E_F)$ of $G[C_1 \cup \cdots C_j \cup W]$ with $V_F \cap \Omega = W$, there is $i \leq \ell$ such that i vertices of F are in $C_1 \cup \cdots C_{j-1} \cup W$ and $\ell - i + |W \cap S_j|$ vertices are in $C_j \cap S_j$.

To compute $\gamma(\ell, j, W, S, C, \Omega)$, we find the blocks (S_j, C_j), $1 \leq j \leq p$, in G, which can be done in time $\mathcal{O}(m)$ and read already computed values $\alpha(\ell - i + |W \cap S_j|, W \cap S_j, S_j, C_j)$ and $\gamma(i, j-1, W, S, C, \Omega)$. Similarly, the values of $\alpha(\ell, W, S, C)$ and $\beta(\ell, W, S, C, \Omega)$ are computable in time $\mathcal{O}(m)$ from the values of the smaller blocks and the values of γ. The total running time required to compute the values of all $\alpha(\ell, W, S, C)$ is $\mathcal{O}(m)$ times the number of different 6-tuple $(\ell, i, W, S, C, \Omega)$ plus the time $\mathcal{O}(n^3(|\Delta_G| + |\Pi_G|))$ required for preprocessing step. The number of good triples (S, C, Ω) is at most $n|\Pi_G|$, and the number of subsets W of size at most $t+1$ is $\mathcal{O}(n^{t+1})$. Thus the total running time required to compute all values $\alpha(\ell, W, S, C)$ is

$$\mathcal{O}(mn^{t+4}(|\Pi_G| + |\Delta_G|)).$$

Now everything is prepared to solve the problem on graph G and to conclude the proof. By Lemma 3.1, if F is an induced subgraph of G of treewidth at most t, there exists a minimal separator S of G, such that $|V_F \cap S| \leq t+1$. We go through all minimal separators, and for each minimal separator S, we try to glue solutions obtained during the first step.

Step 2: Gluing pieces together. Let S be a minimal separator and let $\{C_1, \ldots, C_p\}$ be the vertex sets of the connected components of $G[V \setminus S]$. We put $S_i = N(C_i)$. For every subset $W \subseteq S$ of size at most $t+1$, and integer $0 \leq \ell \leq n$, we define $\delta(\ell, j, W, S) = 1$ if there is an induced ℓ-vertex subgraph $F = (V_F, E_F)$ of $G[W \cup \bigcup_{i=1}^{j} C_i]$ which poses a minimal triangulation TF with $\omega(TF) \leq t+1$, and such that $W = V_F \cap S$ is a clique in TF. If no such graph F exists, we put $\delta(\ell, j, W, S) = 0$. By Lemma 3.1, G has an induced ℓ-vertex subgraph of treewidth at most t if and only if $\delta(\ell, p, W, S) = 1$ for some minimal separator S. Thus computing the value δ for all minimal separators is sufficient for deciding if G has an induced subgraph on ℓ vertices of treewidth at most t.

For every $\ell \geq 0$ and $j = 1$, we have that

$$\delta(\ell, 1, W, S) = \alpha(\ell - |W \setminus S_1|, W \cap S_1, S_1, C_1).$$

For $j > 1$,

$$\delta(\ell, j, W, S) = \begin{cases} 1, & \text{if } \delta(i, j-1, W, S) = 1 \land \alpha(\ell - i + |W \cap S_j|, W \cap S_j, S_j, C_j) = 1, \\ & \text{for some } 1 \leq i \leq \ell, \\ 0, & \text{otherwise.} \end{cases}$$

Like in the case with γ, the correctness of the formula above follows from the fact, that for every ℓ-vertex subgraph $F = (V_F, E_F)$ of $G[C_1 \cup \cdots C_j \cup W]$ with $V_F \cap S = W$, there is $i \leq \ell$ such that i vertices of F are in $C_1 \cup \cdots C_{j-1} \cup W$ and $\ell - i + |W \cap S_j|$ vertices are in $C_j \cap S_j$.

Concerning the time required to perform this step. Like in above, in time $\mathcal{O}(m)$ we can find the connected components $\{C_1, \ldots, C_p\}$ of $G[V \setminus S]$, and the corresponding full blocks (S_i, C_i). Thus the running of this step is proportional to m times the number of 4-tuples (ℓ, j, W, S), and we conclude that this step of the algorithm can be performed in time $\mathcal{O}(mn^{t+3} \cdot |\Delta_G|)$. ∎

4. Induced subgraph isomorphism

The technique described in the previous section with slight modifications can be applied for many different problems. In this section we give an important example of such modification.

Theorem 4.1. *Let G be an n-vertex graph given together with the set Π_G of its potential maximal cliques and the set Δ_G of its minimal separators. Let F be a graph of treewidth t. There is an algorithm checking if G contains an induced subgraph isomorphic to F in time $\mathcal{O}(n^{\mathcal{O}(t)}(|\Delta_G| + |\Pi_G|))$.*

Proof. The proof of the theorem follows the lines of Theorem 3.2 with modifications that are similar to the well known Bodlaender's algorithm for solving the graph isomorphism problem on graphs of bounded treewidth [4]. We outline only the most important differences of such a modification.

The treewidth of F is at most t, and we use the algorithm of Arnborg et.al. [2] to construct a minimal triangulation TF of F such that $\omega(TF) \leq t+1$. The running time of this algorithm is in $\mathcal{O}(n^{t+2})$. The number of maximal cliques and minimal separators in an n-vertex chordal graph is $\mathcal{O}(n)$ [24]. Thus the number of full blocks and good triples in TF is $\mathcal{O}(n)$. We list and keep all these blocks and triples. This can be done in polynomial time.

As in the proof of Theorem 3.2, we perform two steps of dynamic programming. First we run computations over full blocks of G, and then use computed values to glue solutions in minimal separators.

For every full block (S, C) of G, every full block (S_F, C_F) of TF, every subset $W \subseteq S$, where $|W| = |S_F| \leq t+1$, and every bijection $\mu\colon S_F \to W$, we define the value $\alpha(S_F, C_F, W, \mu, S, C)$ to be equal to 1 if there is an injection $\lambda\colon S_F \cup C_F \to W \cup C$ such that $F[S_F \cup C_F]$ is isomorphic to $G[\lambda(S_F \cup C_F)]$, and for every $v \in S_F$, $\lambda(v) = \mu(v)$. Otherwise, we put $\alpha(S_F, C_F, W, \mu, S, C) = 0$. In other words, α is equal to 1, when $G[W \cup C]$ contains a subgraph isomorphic to $F[S_F \cup C_F]$, and moreover, the restriction of the corresponding isomorphic mapping on S_F is exactly μ.

As in Theorem 3.2, to compute $\alpha(S_F, C_F, W, \mu, S, C)$ we run through good triples (S, C, Ω), where Ω is a potential maximal clique, $S \subseteq \Omega \subseteq S \cup C$. For every good triple (S, C, Ω) of G and every good triple (S_F, C_F, Ω_F) of F, for every subset $W \subseteq \Omega$, such that $|W| = |\Omega_F| \leq t+1$, and every bijection $\mu\colon \Omega_F \to W$, we define the function $\beta(S_F, C_F, \Omega_F, W, \mu, S, C, \Omega) \in \{0, 1\}$. We put $\beta(S_F, C_F, \Omega_F, W, \mu, S, C, \Omega) = 1$ if and only if there is an injection $\lambda\colon S_F \cup C_F \to W \cup C$ such that $F[S_F \cup C_F]$ is isomorphic to $G[\lambda(S_F \cup C_F)]$, and for every $v \in \Omega_F$, $\lambda(v) = \mu(v)$. Following the lines of Theorem 3.2, it is possible to show that $\alpha(S_F, C_F, W, \mu, S, C) = 1$ if and only if there exist

- Good triple (S, C, Ω) of G and good triple (S_F, C_F, Ω_F) of F;
- Set W', $W \subseteq W' \subseteq \Omega$;
- Bijection $\mu'\colon \Omega_F \to W'$, $\mu'_{|W}(\cdot) = \mu(\cdot)$

such that $\beta(S_F, C_F, \Omega_F, W', \mu', S, C, \Omega) = 1$.

The main difference with the proof of Theorem 3.2 is in the way we compute β. We compute the values of $\beta(S_F, C_F, \Omega_F, W, \mu, S, C, \Omega)$ from the values of smaller blocks contained in $G[S \setminus \Omega]$. This is done by reducing to the problem of finding a maximum matching in some auxiliary bipartite graph. This step is quite similar to the algorithm of Bodlaender [4] for isomorphism of bounded treewidth graphs. Let $F_1, F_2, \dots, F_p$ be the connected components of the graph $F[C_F \setminus \Omega_F]$. Then the sets $Q_i = N_F(F_i) \subseteq \Omega_F$ are minimal separators and pairs (F_i, Q_i), $1 \leq i \leq p$, are blocks in F. Similarly, for the connected components $G_1, G_2, \dots, G_q$ of $G[C \setminus \Omega]$, we put $S_i = N_G(G_i)$, and define blocks (G_i, S_i), $1 \leq i \leq q$. We construct an auxiliary bipartite graph B with bipartition $X = \{x_1, x_2, \dots, x_p\}$ and $Y = \{y_1, y_2, \dots, y_q\}$. There is an edge $\{x_i, y_j\}$ in B if and only if there is an isomorphic mapping of block (F_i, Q_i) to block (G_j, Q_j) which agrees with μ. But then to decide if blocks (F_i, Q_i) can be mapped to blocked (G_i, S_i) is equivalent to deciding if B has a matching of size p. More formally, $\{x_i, y_j\}$ is an edge in B if and only if there is an injection $\lambda\colon F_i \cup Q_i \to G_j \cup S_j$ such that $F[F_i \cup Q_i]$ is isomorphic to $G[\lambda(F_i \cup Q_i)]$, and for every $v \in Q_i$, $\lambda(v) = \mu(v)$. But such an injection λ exists if and only if $\alpha(F_i, Q_i, W', \mu', G_j, S_j) = 1$, where $W' = \mu(Q_i)$ and $\mu'(\cdot) = \mu_{|Q_i}(\cdot)$. Therefore, to compute the value of β, it is sufficient to run through the already computed values of α of smaller blocks, construct an auxiliary graph and find if this graph contains a matching of specific size.

Finally, as in Theorem 3.2, after all values α are computed, we run through all minimal separators of G and for each minimal separator S, we try to glue solutions obtained for all blocks attached to this separator. Here again, we need only the values of α computed for all such blocks and reduce the problem to bipartite matchings. The running time of the algorithm is up to multiplicative polynomial factor equal to the number of states of the dynamic programming. To compute the values of α and β, we run through all potential maximal cliques, blocks, and good triples of TF and G, which is $n^{\mathcal{O}(1)}|\Pi_G|$. For every pair of blocks or triples, we run through all subsets W of size at most $t+1$, which is $\mathcal{O}(n^{t+1})$, and through all mappings between sets of cardinality at most $t+1$, which is $\mathcal{O}((t+1)^{t+1})$. Finally, we run through all minimal separators. Thus the total running time of the algorithm is $\mathcal{O}(n^{\mathcal{O}(t)}(|\Delta_G| + |\Pi_G|))$. The proof of the correctness of the algorithm follows the lines of Theorem 3.2, and we omit it here. ■

Let us also remark that with a standard bookkeeping, the algorithm of Theorem 4.1 can also output a subgraph of G isomorphic to F.

5. Enumerating potential maximal cliques

In this section we show that all potential maximal cliques of graph $G = (V, E)$ can be enumerated by making use of connected vertex sets with special restrictions. This approach represents a significant simplification over previous algorithms for listing potential maximal cliques [12, 13]. More precisely, we show that for every potential maximal clique Ω there exists a vertex set $Z \subset V$ and a vertex $z \in Z$ such that

- $|Z| - 1 \leq (2/3)(n - |\Omega|)$,
- $G[Z]$ is connected,
- $\Omega = N(Z \setminus \{z\})$ or $\Omega = N(Z) \cup \{z\}$.

As far as we obtain such a classification, the enumeration algorithm is extremely simple: For each vertex $z \in V$ enumerate every connected vertex set Z containing z where $|Z| - 1 \leq 2|V \setminus N[Z - \{z\}]|$. (In other words we test for each connected vertex set Z containing z, where at least $\frac{|Z|-1}{2}$ vertices are not contained in $N[Z \setminus \{z\}]$.) For each of these subsets, we run the algorithm of Bouchitté and Todinca from [5] to check if $N(Z \setminus \{z\})$ or $N(Z) \cup \{z\}$ is a potential maximal clique. The algorithm of Bouchitté and Todinca checks in $\mathcal{O}(nm)$ time if a vertex set Ω is a potential maximal clique. This is a significant simplification comparing to previous enumeration algorithms [12, 13] avoiding complications with different treatments of nice and (not) nice potential maximal cliques.

We proceed with a sequence of technical lemmas. For a potential maximal clique Ω and a vertex $x \in \Omega$ we define by D_x the vertex sets of all connected components C of $G[V \setminus \Omega]$ with $x \in N(C)$.

Lemma 5.1. *Let Ω be a potential maximal clique of $G = (V, E)$, and let $\{x, y\}$ be an edge of $G[\Omega]$ such that Ω is not a potential maximal clique in $G \setminus \{x, y\}$. Then there is $Z \subseteq V$ and $z \in Z$, such that*

- *$\Omega = N(Z) \cup \{z\}$,*
- *$G[Z]$ is connected, and*
- *$|Z| - 1 \leq (1/2)(n - |\Omega|)$.*

Corollary 5.2. *Let Ω be a potential maximal clique of $G = (V, E)$, such that Ω is a potential maximal clique in $G\backslash\{x, y\}$ for every edge $\{x, y\}$ of $G[\Omega]$. Then $N(D_x) = \Omega$ for every vertex $x \in \Omega$.*

Let $\mathcal{C}$ be the set of connected components of $G[V \setminus \Omega]$ with the following two properties: For each connected component $C \in \mathcal{C}$ there exists a pair of vertices $x, y \in \Omega$ such that C is the unique component from $\mathcal{C}$ with $x, y \in N(C)$, and for each pair of vertices $x, y \in \Omega$ there exists a connected component $C \in \mathcal{C}$ such that $x, y \in N(C)$. Let W be the vertex set of $\mathcal{C}$, we refer to the graph $G' = G[\Omega \cup W]$ as to a *reduced graph for* Ω. In other words $\mathcal{C}$ is an inclusion minimal witness for Ω being a potential maximal clique of G, by only using connected components of $G[V \setminus \Omega]$. The set $\mathcal{C}$ can be constructed by the following procedure which is repeated recursively if possible: If there exists a connected component C of $G[V \setminus \Omega]$ such that for each pair $x, y \in N(C)$ there is a connected component $C' \neq C$ in $G[V \setminus \Omega]$ such that $x, y \in N(C')$, then remove C from the graph.

Lemma 5.3. *Let Ω be a potential maximal clique of $G = (V, E)$ such that Ω is also a potential maximal clique in $G\backslash\{x, y\}$ for every edge $\{x, y\}$ of $G[\Omega]$, and where $G' = G[\Omega \cup W]$ contains at least 4 connected components. Then there is $Z \subset V$ and $z \in Z$ such that*

- $\Omega = N(Z \setminus \{z\})$,
- *$G[Z]$ is connected, and*
- $|Z| - 1 \leq (3/5)(n - |\Omega|)$.

The following characterization is used in the new algorithm enumerating potential maximal cliques.

Lemma 5.4. *For every potential maximal clique Ω of $G = (V, E)$, there exists a vertex set $Z \subseteq V$ and $z \in Z$ such that*

- $|Z| - 1 \leq (2/3)(n - |\Omega|)$,
- *$G[Z]$ is connected, and*
- *$\Omega = N(Z \setminus \{z\})$ or $\Omega = N(Z) \cup \{z\}$.*

Let us remark that Lemma 5.4 yields a simple algorithm enumerating potential maximal cliques. We just connected vertex sets Z of bounded size and check if either $N(Z \setminus \{z\})$ or $N(Z) \cup \{z\}$ is a potential maximal clique. The enumeration of such connected vertex sets can be done in time $\mathcal{O}(n^2 \cdot 1.7549^n)$ [13] and checking if a set is a potential maximal clique in $\mathcal{O}(nm)$ time [5].

In what follows we improve (slightly) the running time of the algorithm. The improvement is based on the previous lemmata. The proof gain by exploiting the fact that the most time consuming case is when there are exactly three connected components in the reduced graph.

Theorem 5.5. *All potential maximal cliques of an n-vertex graph can be enumerated in time $\mathcal{O}(1.734601^n)$.*

We need the following results.

Theorem 5.6 (Berry, Bordat, and Cogis [3])**.** *There is an algorithm listing all minimal separators of an input graph G in $\mathcal{O}(n^3|\Delta_G|)$ time.*

Theorem 5.7 (Fomin and Villanger [13])**.** *Every n-vertex graph has $\mathcal{O}(1.6181^n)$ minimal separators.*

Putting together Theorems 3.2, 5.5, 5.6, and 5.7, we arrive at the following corollary.

Corollary 5.8. *For every $t \geq 0$, a maximum induced subgraph of treewidth at most t in an n-vertex graph G can be found in time $\mathcal{O}(1.734601^n \cdot n^{\mathcal{O}(t)})$.*

Similarly, by Theorem 4.1, we have the following corollary.

Corollary 5.9. *For every $t \geq 0$ and graph F of treewidth t, checking if an n-vertex graph G contains an induced subgraph isomorphic to F (and finding one if such exist) can be done in time $\mathcal{O}(1.734601^n \cdot n^{\mathcal{O}(t)})$.*

Let us remark that the treewidth of an n-vertex planar, and more generally, graph excluding some fixed graph as a minor, is $\mathcal{O}(\sqrt{n})$ [1]. Therefore, if F is a graph excluding some fixed graph as a minor, deciding if G has induced subgraph isomorphic to F can be done in time $1.734601^{n+o(n)}$.

6. Conclusion and open questions

In this paper we have shown how the theory of minimal triangulations can be used to obtain moderate exponential algorithms for a number of problems about induced subgraphs. With some modifications our technique can be used for different problems of the same flavor, like finding a maximum connected induced subgraph of small treewidth. It would be interesting to see if Theorem 3.2 can be extended for finding maximum induced subgraphs with other specific properties like being planar or excluding some h-vertex graph H as a minor.

Another very interesting question is, how many potential maximal cliques can be in an n-vertex graph? Theorem 5.5 says that roughly at most 1.734601^n. How tight is this bound? There are graphs with roughly $3^{n/3} \approx 1.442^n$ potential maximal cliques [12]. Let us remind that by the classical result of Moon and Moser [19] (see also Miller and Muller [18]) that the number of maximal cliques in a graph on n vertices is at most $3^{n/3}$. Can it be that the right upper bound on the number of potential maximal cliques is also roughly $3^{n/3}$? By Theorem 3.2, this would yield a dramatic improvement for many moderate exponential algorithms.

References

[1] N. Alon, P. Seymour, and R. Thomas, *A separator theorem for nonplanar graphs*, J. Amer. Math. Soc., 3 (1990), pp. 801–808.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 277–284.

[3] A. Berry, J. P. Bordat, and O. Cogis, *Generating all the minimal separators of a graph*, Int. J. Found. Comput. Sci., 11 (2000), pp. 397–403.

[4] H. L. Bodlaender, *Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees*, J. Algorithms, 11 (1990), pp. 631–643.

[5] V. Bouchitté and I. Todinca, *Treewidth and minimum fill-in: Grouping the minimal separators*, SIAM J. Comput., 31 (2001), pp. 212–232.

[6] ———, *Listing all potential maximal cliques of a graph*, Theor. Comput. Sci., 276 (2002), pp. 17–32.

[7] J. Chen, I. A. Kanj, and G. Xia, *Labeled search trees and amortized analysis: improved upper bounds for NP-hard problems*, Algorithmica, 43 (2005), pp. 245–273.

[8] M. Davis and H. Putnam, *A computing procedure for quantification theory*, J. Assoc. Comput. Mach., 7 (1960), pp. 201–215.

[9] F. V. Fomin, S. Gaspers, and A. V. Pyatkin, *Finding a minimum feedback vertex set in time $O(1.7548^n)$*, in Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC 2006), vol. 4169 of Lecture Notes in Comput. Sci., Springer, Berlin, 2006, pp. 184–191.

[10] F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon, *On the minimum feedback vertex set problem: Exact and enumeration algorithms*, Algorithmica, 52 (2008), pp. 293–307.

[11] F. V. Fomin, F. Grandoni, and D. Kratsch, *Measure and conquer: A simple $O(2^{0.288\,n})$ independent set algorithm*, in 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006), New York, 2006, ACM and SIAM, pp. 18–25.

[12] F. V. Fomin, D. Kratsch, I. Todinca, and Y. Villanger, *Exact algorithms for treewidth and minimum fill-in*, SIAM J. Comput., 38 (2008), pp. 1058–1079.

[13] F. V. Fomin and Y. Villanger, *Treewidth computation and extremal combinatorics*, in Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP 2008), LNCS, Springer, 2008, pp. 210–221.

[14] M. Fürer, *A faster algorithm for finding maximum independent sets in sparse graphs*, in Proceedings of the 7th Latin American Theoretical Informatics Symposium (LATIN 2006), vol. 3887 of Lecture Notes in Comput. Sci., Springer-Verlag, Berlin, 2006, pp. 491–501.

[15] S. Gupta, V. Raman, and S. Saurabh, *Fast exponential algorithms for maximum r-regular induced subgraph problems*, in Proceedings of the 26th International Conference Foundations of Software Technology and Theoretical Computer Science, (FSTTCS 2006), LNCS, Springer-Verlag, 2006, pp. 139–151.

[16] P. Heggernes, *Minimal triangulations of graphs: A survey*, Discrete Mathematics, 306 (2006), pp. 297–317.

[17] T. Jian, *An $O(2^{0.304n})$ algorithm for solving maximum independent set problem*, IEEE Trans. Computers, 35 (1986), pp. 847–851.

[18] R. E. Miller and D. E. Muller, *A problem of maximum consistent subsets*, IBM Research Rep. RC-240, J.T. Watson Research Center, Yorktown Heights, New York, USA, 1960.

[19] J. W. Moon and L. Moser, *On cliques in graphs*, Israel Journal of Mathematics, 3 (1965), pp. 23–28.

[20] I. Razgon, *Exact computation of maximum induced forest*, in Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006), vol. 4059 of Lecture Notes in Comput. Sci., Berlin, 2006, Springer, pp. 160–171.

[21] ———, *A faster solving of the maximum independent set problem for graphs with maximal degree 3*, in Proceedings of the Second ACiD Workshop (ACiD 2006), vol. 7 of Texts in Algorithmics, King's College, London, 2006, pp. 131–142.

[22] N. Robertson and P. D. Seymour, *Graph minors. II. Algorithmic aspects of tree-width*, Journal of Algorithms, 7 (1986), pp. 309–322.

[23] J. M. Robson, *Algorithms for maximum independent sets*, Journal of Algorithms, 7 (1986), pp. 425–440.

[24] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.

[25] R. E. Tarjan and A. E. Trojanowski, *Finding a maximum independent set*, SIAM Journal on Computing, 6 (1977), pp. 537–546.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 395-404
www.stacs-conf.org

INSEPARABILITY AND STRONG HYPOTHESES FOR DISJOINT NP PAIRS

LANCE FORTNOW[1] AND JACK H. LUTZ[2] AND ELVIRA MAYORDOMO[3]

[1] Northwestern University, EECS Department, Evanston, Illinois, USA.
E-mail address: `fortnow@eecs.northwestern.edu`

[2] Department of Computer Science, Iowa State University, Ames, IA 50011 USA.
E-mail address: `lutz@cs.iastate.edu`

[2] Departamento de Informática e Ingeniería de Sistemas, Instituto de Investigación en Ingeniería de Aragón, María de Luna 1, Universidad de Zaragoza, 50018 Zaragoza, SPAIN.
E-mail address: `elvira\at\unizar.es`

Abstract. This paper investigates the existence of inseparable disjoint pairs of NP languages and related strong hypotheses in computational complexity. Our main theorem says that, if NP does not have measure 0 in EXP, then there exist disjoint pairs of NP languages that are P-inseparable, in fact TIME(2(n k))-inseparable. We also relate these conditions to strong hypotheses concerning randomness and genericity of disjoint pairs.

1. Introduction

The main objective of complexity theory is to assess the intrinsic difficulties of naturally arising computational problems. It is often the case that a problem of interest can be formulated as a decision problem, or else associated with a decision problem of the same complexity, so much of complexity theory is focused on decision problems. Nevertheless, other types of problems also require investigation.

This paper concerns *promise problems*, a natural generalization of decision problems introduced by Even, Selman, and Yacobi [7]. A decision problem can be formulated as a set $A \subseteq \{0,1\}^*$, where a solution of this problem is an algorithm, circuit, or other device that *decides* A, i.e., tells whether or not an arbitrary input $x \in \{0,1\}^*$ is an element of A. In contrast, a promise problem is formulated as an ordered pair (A, B) of disjoint sets $A, B \subseteq \{0,1\}^*$, where a solution is an algorithm or other device that decides *any* set $S \subseteq \{0,1\}^*$ such that $A \subseteq S$ and $B \cap S = \emptyset$. Such a set S is called a *separator* of the disjoint pair (A, B). Intuitively, if we are promised that every input will be an element of

1998 ACM Subject Classification: F.1.3.

Key words and phrases: Computational Complexity, Disjoint NP-pairs, Resource-Bounded Measure, Genericity.

Thanks: Fortnow's research supported in part by NSF grants CCF-0829754 and DMS-0652521. Lutz's research supported in part by National Science Foundation Grants 0344187, 0652569, and 0728806. Mayordomo's research supported in part by Spanish Government MICINN Project TIN2008-06582-C03-02.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2471

$A \cup B$, then a separator of (A, B) enables us to distinguish inputs in A from inputs in B. Since each decision problem A is clearly equivalent to the promise problem (A, A^c), where $A^c = \{0,1\}^* - A$ is the complement of A, promise problems are, indeed, a generalization of decision problems.

A *disjoint NP pair* is a promise problem (A, B) in which $A, B \in \mathrm{NP}$. Disjoint NP pairs were first investigated by Selman and others in connection with public key cryptosystems [7, 15, 26, 17]. They were later investigated by Razborov [25] as a setting in which to prove the independence of complexity-theoretic conjectures from theories of bounded arithmetic. In this same paper, Razborov established a fundamental connection between disjoint NP pairs and propositional proof systems. Propositional proof systems had been used by Cook and Reckhow [6] to characterize the NP versus co-NP problem. Razborov [25] showed that each propositional proof system has associated with it a canonical disjoint NP pair and that important questions about propositional proof systems are thereby closely related to natural questions about disjoint NP pairs. This connection with propositional proof systems has motivated more recent work on disjoint NP pairs by Glaßer, Selman, Sengupta, and Zhang [10, 9, 12, 13]. It is now known that the degree structure of propositional proof systems under the natural notion of proof simulation is identical to the degree structure of disjoint NP pairs under reducibility of separators [12]. Much of this recent work is surveyed in [11]. Goldreich [14] gives a recent survey of promise problems in general.

Our specific interest in this paper is the existence of disjoint NP pairs that are P-inseparable, or even $\mathrm{TIME}(2^{n^k})$-inseparable. As the terminology suggests, if $\mathcal{C}$ is a class of decision problems, then a disjoint pair is *$\mathcal{C}$-inseparable* if it has no separator in $\mathcal{C}$. The existence of P-inseparable disjoint NP pairs is a strong hypothesis in the sense that (1) it clearly implies $\mathrm{P} \neq \mathrm{NP}$, and (2) the converse implication is not known (and fails relative to some oracles [17]). It is clear that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ implies the existence of P-inseparable disjoint NP pairs, and Grollmann and Selman [15] proved that $\mathrm{P} \neq \mathrm{UP}$ also implies the existence of P-inseparable disjoint NP pairs.

The hypothesis that NP is a non-measure 0 subset of EXP, written $\mu(\mathrm{NP} \mid \mathrm{EXP}) \neq 0$, is a strong hypothesis in the above sense. This hypothesis has been shown to have many consequences not known to follow from more traditional hypotheses such as $\mathrm{P} \neq \mathrm{NP}$ or the separation of the polynomial-time hierarchy into infinitely many levels. Each of these known consequences has resolved some pre-existing complexity-theoretic question in the way that agreed with the conjecture of most experts. This explanatory power of the $\mu(\mathrm{NP} \mid \mathrm{EXP}) \neq 0$ hypothesis is discussed in the early survey papers [23, 2, 24] and is further substantiated by more recent papers listed at [16] (and too numerous to discuss here). In several instances, the discovery that $\mu(\mathrm{NP} \mid \mathrm{EXP}) \neq 0$ implies some plausible conclusion has led to subsequent work deriving the same conclusion from some weaker hypothesis, thereby further illuminating the relationships among strong hypotheses.

Our main theorem states that, if NP does not have measure zero in EXP, then, for every positive integer k, there exist disjoint NP pairs that are $\mathrm{TIME}(2^{n^k})$-inseparable. Such pairs are *a fortiori* P-inseparable, but the conclusion of our main theorem actually gives *exponential* lower bounds on the inseparability of some disjoint NP pairs. These are the lower bounds that most experts conjecture to be true, even though an unconditional proof of such bounds may be long in coming.

The proof of our main theorem combines known closure properties of NP with the randomness that the $\mu(\mathrm{NP} \mid \mathrm{EXP}) \neq 0$ hypothesis implies must be present in NP to give an explicit construction of a disjoint NP pair that is $\mathrm{TIME}(2^{n^k})$-inseparable. (Technically, this is an overstatement. The last step of the "construction" is the removal of a finite set whose existence we prove, but which we do not construct.) The details are perhaps involved, but we preface the proof with an intuitive motivation for the approach.

We also investigate the relationships between the two strong hypotheses in our main theorem (i.e., its hypothesis and its conclusion) and strong hypotheses involving the existence of disjoint NP pairs with randomness and genericity properties. Roughly speaking (i.e., omitting quantitative parameters), we show that the existence of disjoint NP pairs that are random implies both the $\mu(\mathrm{NP} \mid \mathrm{EXP}) \neq 0$ hypothesis and the existence of disjoint NP pairs that are generic in the sense of Ambos-Spies, Fleischhack, and Huwig [1]. We also show that the existence of such generic pairs implies the existence of disjoint NP pairs that are $\mathrm{TIME}(2^{n^k})$-inseparable. Taken together, these results give the four implications at the top of Figure 1. (The four implications at the bottom are well known.) We prove that three of these implications cannot be reversed by relativizable techniques, and we conjecture that this also holds for the remaining implication.

2. Preliminaries

We write $\mathbb{N}$ for the set of nonnegative integers and $\mathbb{Z}^+$ for the set of (strictly) positive integers. The *Boolean value* of an assertion ϕ is $[\![\phi]\!] =$ *if* ϕ *then 1 else 0.* All logarithms here are base-2.

We write λ for the empty string, $|w|$ for the length of a string w, and $s_0, s_1, s_2, \ldots$ for the standard enumeration of $\{0,1\}^*$. The index of a string x is the value $\mathrm{ind}(x) \in \mathbb{N}$ such that $s_{\mathrm{ind}(x)} = x$. We write $\mathrm{next}(x)$ for the string following x in the standard enumeration, i.e., $\mathrm{next}(s_n) = s_{n+1}$. More generally, for $k \in \mathbb{N}$, we write next^k for the k-fold composition of next with itself, so that $\mathrm{next}^k(s_n) = s_{n+k}$.

A *Boolean function* is a function $f : \{0,1\}^m \to \{0,1\}$ for some $m \in \mathbb{N}$. The *support* of such a function f is $\mathrm{supp}(f) = \left\{ x \in \{0,1\}^m \,\middle|\, f(x) = 1 \right\}$.

We write $w[i]$ for the i^{th} symbol in a string w and $w[i..j]$ for the string consisting of the i^{th} through j^{th} symbols. The leftmost symbol of w is $w[0]$, so that $w = w[0..|w|-1]$. For (infinite) sequences $S \in \Sigma^\infty$, the notations $S[i]$ and $S[i..j]$ are defined similarly. A string $w \in \Sigma^*$ is a prefix of a string or sequence $x \in \Sigma^* \cup \Sigma^\infty$, and we write $w \sqsubseteq x$, if there is a string or sequence $y \in \Sigma^* \cup \Sigma^\infty$ such that $wy = x$. A *language*, or *decision problem*, is a set $A \subseteq \{0,1\}^*$. We identify each language A with the sequence $A \in \{0,1\}^\infty$ defined by $A[n] = [\![s_n \in A]\!]$ for all $n \in \mathbb{N}$. If A is a language, then expressions like $\lim_{w \to A} f(w)$ refer to prefixes $w \sqsubseteq A$, e.g., $\lim_{w \to A} f(w) = \lim_{n \to \infty} f(A[0..n-1])$.

A *martingale* is a function $d : \{0,1\}^* \to [0,\infty)$ satisfying

$$d(w) = \frac{d(w0) + d(w1)}{2} \tag{2.1}$$

for all $w \in \{0,1\}^*$. Intuitively, d is a *strategy for betting* on the successive bits of a sequence $S \in \{0,1\}^\infty$: The quantity $d(w)$ is the amount of money that the gambler using this strategy has after $|w|$ bets if $w \sqsubseteq S$. Condition (2.1) says that the payoffs are fair.

A martingale d *succeeds* on a language $A \subseteq \{0,1\}^*$, and we write $A \in S^\infty[d]$, if $\limsup_{w \to A} d(w) = \infty$. If $t : \mathbb{N} \to \mathbb{N}$, then a martingale d is (*exactly*) $t(n)$-*computable* if its values are rational and there is an algorithm that computes each $d(w)$ in $t(|w|)$ time. A martingale is p-*computable* if it is n^k-computable for some $k \in \mathbb{N}$, and it is p_2-computable if it is $2^{(\log n)^k}$-computable for some $k \in \mathbb{N}$.

Definition 2.1. [22] Let X be a set of languages, and let R be a language.

(1) X has p-*measure 0*, and we write $\mu_{\mathrm{p}}(X) = 0$, if there is a p-computable martingale d such that $X \subseteq S^\infty[d]$. The condition $\mu_{\mathrm{p}_2}(X) = 0$ is defined analogously.
(2) X has *measure 0 in* EXP, and we write $\mu(X \mid \mathrm{EXP}) = 0$, if $\mu_{\mathrm{p}_2}(X \cap \mathrm{EXP}) = 0$.
(3) R is p-*random* if $\mu_{\mathrm{p}}(\{R\}) \neq 0$, i.e., if there is no p-computable martingale that succeeds on R. Similarly, R is $t(n)$-*random* if no $t(n)$- computable martingale succeeds on R.

It is well known that these definitions impose a nontrivial measure structure on EXP [22]. For example, $\mu(\mathrm{EXP} \mid \mathrm{EXP}) \neq 0$.

We use the following fact in our arguments.

Lemma 2.2. [3, 18] *The following five conditions are equivalent.*

(1) $\mu(\mathrm{NP} \mid \mathrm{EXP}) \neq 0$.
(2) $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$.
(3) $\mu_{\mathrm{p}_2}(\mathrm{NP}) \neq 0$.
(4) There exists a p-random language $R \in \mathrm{NP}$.
(5) For every $k \geq 2$, *there exists an* $2^{\log n^k}$*-random language* $R \in \mathrm{NP}$.

Finally, we note that $\mu(\mathrm{P} \mid \mathrm{EXP}) = 0$ [22], so $\mu(\mathrm{NP} \mid \mathrm{EXP}) \neq 0$ implies $\mathrm{P} \neq \mathrm{NP}$.

3. Inseparable Disjoint NP Pairs and the Measure of NP

This section presents our main theorem, which says that, if NP does not have measure 0 in EXP, then there are disjoint NP pairs that are P-inseparable. In fact, for each $k \in \mathbb{N}$, there is a disjoint NP pair that is $\mathrm{TIME}(2^{n^k})$-inseparable.

It is convenient for our arguments to use a slight variant of the separability notion.

Definition 3.1. Let (A, B) be a pair of (not necessarily disjoint) languages, and let $\mathcal{C}$ be a class of languages.

(1) A language $S \subseteq \{0,1\}^*$ *almost separates* (A, B) if there is a finite set $D \subseteq \{0,1\}^*$ such that S separates $(A - D, B - D)$.
(2) We say that (A, B) is $\mathcal{C}$-*almost separable* if there is a language $S \in \mathcal{C}$ that almost separates (A, B).

Observation 3.2. If a pair (A, B) is not $\mathcal{C}$-almost separable, then $(A - D, B - D)$ is $\mathcal{C}$-inseparable for every finite set D.

Before proving our main theorem, we sketch the intuitive idea of the proof. We want to construct a disjoint NP pair (A, B) that is P-inseparable. Our hypothesis, that NP does not have measure 0 in EXP, implies that NP contains a language R that is p-random. Since we are being intuitive, we ignore the subtleties of p-randomness and regard R as a sequence of independent, fair coin tosses (with the n^{th} toss heads iff $s_n \in R$) that just happens to be in NP. If we use these coins to randomly put strings in A or B but not both, we can count on the randomness to thwart any would-be separator in P.

The challenge here is that, if we are to deduce $A, B \in \mathrm{NP}$ from $R \in \mathrm{NP}$, we must make the conditions "$s_n \in A$" and "$s_n \in B$" depend on the coin tosses in a *monotone* way; i.e., adding a string to R must not move a string out of A or out of B.

This monotonicity restriction might at first seem to prevent us from ensuring that A and B are disjoint. However, this is not the case. Suppose that we decide membership of the n^{th} string s_n in A and B in the following manner. We toss $2\log n$ independent coins. If the first $\log n$ tosses all come up heads, we put s_n in A. If the second $\log n$ tosses all come up heads, we put s_n in B. If our coin tosses are taken from R, which is in NP, then A and B will be in NP. Each string s_n will be in A with probability $\frac{1}{n}$, in B with probability $\frac{1}{n}$, and in $A \cap B$ with probability $\frac{1}{n^2}$. Since $\sum_{n=1}^{\infty} \frac{1}{n}$ diverges and $\sum_{n=1}^{\infty} \frac{1}{n^2}$ converges, the first and second Borel-Cantelli lemmas tell us that A and B are infinite and $A \cap B$ is finite. Since $A \cap B$ is finite, we can subtract it from A and B, leaving two disjoint NP languages that are, by the randomness of the construction, P-inseparable.

What prevents this intuitive argument from being a proof sketch is the fact that the language R is not truly random, but only p-random. The proof that $A \cap B$ is finite thus becomes problematic. There is a resource-bounded extension of the first Borel-Cantelli lemma [22] that works for p-random sequences, but this extension requires the relevant sum of probabilities to be p-convergent, i.e., to converge much more quickly than $\sum_{n=1}^{\infty} \frac{1}{n^2}$.

Fortunately, in this particular instance, we can achieve our objective without p-convergence or the (classical or resource-bounded) Borel-Cantelli lemmas. We do this by modifying the above construction. Instead of putting the n^{th} string into each language with probability $\frac{1}{n}$, we put each string x into each of A and B with probability $2^{-|x|}$ so that x is in $A \cap B$ with probability $2^{-2|x|}$. By the Cauchy condensation test, the relevant series have the same convergence behavior as those in our intuitive argument, but we can now replace slow approximations of tails of $\sum_{n=1}^{\infty} \frac{1}{n^2}$ with fast and exact computations of geometric series.

We now turn to the details.

Construction 3.3. (1) Define the functions $u, v : \{0,1\}^* \to \{0,1\}^*$ by the recursion

$$\begin{aligned} &u(\lambda) = \lambda, \\ &v(x) = \mathrm{next}^{|x|}(u(x)), \\ &u(\mathrm{next}(x) = \mathrm{next}^{|x|}(v(x)). \end{aligned}$$

(2) For each $x \in \{0,1\}^*$, define the intervals

$$I_x = [u(x), v(x)), \;\; J_x = [v(x), u(next(x))).$$

(3) For each $R \subseteq \{0,1\}^*$, define the languages

$$\begin{aligned} &A^+(R) = \left\{ x \,\middle|\, I_x \subseteq R \right\}, \;\; B^+(R) = \left\{ x \,\middle|\, J_x \subseteq R \right\}, \\ &A(R) = A^+(R) - B^+(R), \;\; B(R) = B^+(R) - A^+(R). \end{aligned}$$

Note that each $|I_x| = |J_x| = |x|$. Also, $I_\lambda = J_\lambda = \emptyset$ (so $\lambda \in A^+(R) \cap B^+(R)$), and

$$I_0 < J_0 < I_1 < J_1 < I_{00} < J_{00} < I_{01} < \ldots,$$

with these intervals covering all of $\{0,1\}^*$.

A routine witness argument gives the following.

Observation 3.4. (1) If $R \in \mathrm{NP}$, then $A^+(R), B^+(R) \in \mathrm{NP}$.

(2) If $R \in \mathrm{NP}$ and $|A^+(R) \cap B^+(R)| < \infty$, then $(A(R), B(R))$ is a disjoint NP pair.

We now prove two lemmas about Construction 3.3.

Lemma 3.5. *Let $k \in \mathbb{N}$. If $R \subseteq \{0,1\}^*$ is $2^{(\log n)^{k+2}}$- random, then $(A^+(R), B^+(R))$ is not* $\mathrm{TIME}(2^{n^k})$*-almost separable.*

Lemma 3.6. *If $R \subseteq \{0,1\}^*$ is p-random, then $|A^+(R) \cap B^+(R)| < \infty$.*

We now have what we need to prove our main result.

Theorem 3.7. *(main theorem) If NP does not have measure 0 in EXP, then, for every $k \in \mathbb{Z}^+$, there is a disjoint NP pair that is* $\mathrm{TIME}(2^{n^k})$*-inseparable, hence certainly P-inseparable.*

Proof. Assume that $\mu(\mathrm{NP} \mid \mathrm{EXP}) \neq 0$, and let $k \in \mathbb{N}$. Then, by Lemma 2.2, there is a $2^{(\log n)^{k+2}}$-random language $R \in \mathrm{NP}$. By Lemma 3.5, the pair $(A^+(R), B^+(R))$ is not $\mathrm{TIME}(2^{n^k})$-almost separable. Since R is certainly p-random, Lemma 3.6 tells us that $|A^+(R) \cap B^+(R)| < \infty$. It follows by Observation 3.4 that $(A(R), B(R))$ is a disjoint NP pair, and it follows by Observation 3.2 that $(A(R), B(R))$ is $\mathrm{TIME}(2^{n^k})$-inseparable. ■

4. Genericity and Measure of Disjoint NP Pairs

In this section we introduce the natural notions of resource-bounded measure and genericity for disjoint pairs and relate them to the existence of P-inseparable pairs in NP. We compare the different strength hypothesis on the measure and genericity of NP and disjNP establishing all the relations in Figure 1.

Notation. Each disjoint pair (A, B) will be coded as an infinite sequence $T \in \{-1, 0, 1\}^\infty$ defined by

$$T[n] = \begin{cases} 1 & \text{if } s_n \in A \\ -1 & \text{if } s_n \in B \\ 0 & \text{if } s_n \notin A \cup B \end{cases}$$

We identify each disjoint pair with the corresponding sequence.

Resource-bounded genericity for disjoint pairs is the natural extension of the concept introduced for languages by Ambos-Spies, Fleischhack and Huwig [1].

Definition 4.1. A *condition* C is a set $C \subseteq \{-1, 0, 1\}^*$. A $t(n)$-condition is a condition $C \in \mathrm{DTIME}(t(n))$. A condition C is *dense along a pair* (A, B) if there are infinitely many $n \in \mathbb{N}$ such that $(A, B)[0..n-1]i \in C$ for some $i \in \{-1, 0, 1\}$. A pair (A, B) *meets* a condition C if $(A, B)[0..n-1] \in C$ for some n. A pair (A, B) is $t(n)$*-generic* if (A, B) meets every $t(n)$-condition that is dense along (A, B).

We first prove that generic pairs are inseparable.

Theorem 4.2. *Every $t(\log n)$-generic disjoint pair is* $\mathrm{TIME}(t(n))$*-inseparable.*

We can now relate genericity in disjNP and inseparable pairs as follows.

Corollary 4.3. *If disjNP contains a $2^{(\log n)^k}$-generic pair for every $k \in \mathbb{N}$, then disjNP contains a* $\mathrm{TIME}(2^{n^k})$*-inseparable pair for every $k \in \mathbb{N}$.*

Resource-bounded measure on classes of disjoint pairs is the natural extension of the concept introduced for languages by Lutz [22], and is defined by using martingales on a three-symbol alphabet as follows.

Definition 4.4. (1) A *pair martingale* is a function $d : \{-1,0,1\}^* \to [0,\infty)$ such that for every $w \in \{-1,0,1\}^*$

$$d(w) = \frac{1}{4}d(w0) + \frac{3}{8}d(w1) + \frac{3}{8}d(w(-1)).$$

(2) A pair martingale d *succeeds on a pair* (A,B) if $\limsup_{w\to(A,B)} d(w) = \infty$.

(3) A pair martingale d *succeeds on a class of pairs* $X \subseteq \{-1,0,1\}^\infty$ if it succeeds on each $(A,B) \in X$.

Our intuitive rationale for the coefficients in part 1 of this definition is the following. We toss one fair coin to decide whether $s_{|w|} \in A$ and another to decide whether $s_{|w|} \in B$. If both coins come up heads, we toss a third coin to break the tie. The reader may feel that some other coefficients, such as $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$ are more natural here. Fortunately, a routine extension of the main theorem of [5] shows that the value of $\mu(\text{disjNP} \mid \text{disjEXP})$ will be the same for *any* choice of three positive coefficients summing to 1.

When restricting martingales to those computable within a certain resource bound, we obtain a resource-bounded measure that is useful within a complexity class. Here we are interested in the class of disjoint EXP pairs, disjEXP.

Definition 4.5. (1) Let p_2 be the class of functions that can be computed in time $2^{(\log n)^{O(1)}}$.

(2) A class of pairs $X \subseteq \{-1,0,1\}^\infty$ has p_2*-measure 0* if there is a martingale $d \in \text{p}_2$ that succeeds on X.

(3) $X \subseteq \{-1,0,1\}^\infty$ has p_2*-measure 1* if X^c has p_2-measure 0.

(4) A class of pairs $X \subseteq \{-1,0,1\}^\infty$ has *measure 0 in disjEXP*, denoted $\mu(X \mid \text{disjEXP}) = 0$, if $X \cap \text{disjEXP}$ has p_2-measure 0.

(5) $X \subseteq \{-1,0,1\}^\infty$ has *measure 1 in disjEXP* if X^c has measure 0 in disjEXP.

It is easy to verify that p_2-measure is nontrivial on disjEXP (as proven for languages in [22]).

In the following we consider the hypothesis that disjNP does not have measure 0 in disjEXP (written $\mu(\text{disjNP} \mid \text{disjEXP}) \neq 0$). We start by proving that this hypothesis is at least as strong as the well studied $\mu(\text{NP} \mid \text{EXP}) \neq 0$ hypothesis.

Theorem 4.6. *If* $\mu(\text{disjNP} \mid \text{disjEXP}) \neq 0$ *then* $\mu(\text{NP} \mid \text{EXP}) \neq 0$.

We finish by relating measure and genericity for disjoint pairs.

Theorem 4.7. *If* $\mu(\text{disjNP} \mid \text{disjEXP}) \neq 0$, *then disjNP contains a* $2^{(\log n)^k}$*-generic pair for every* $k \in \mathbb{N}$.

5. Oracle Results

All the techniques in this and related papers relativize, that is they hold when all machines involved have access to the same oracle A. In this section we give relativized worlds where the converses of most of the results in this paper, as expressed in Figure 1, do not hold. Since the implications trivially all hold in any relativized world where P = NP [4], one cannot use relativizable techniques to settle these converses.

We'll work our way from the bottom up of Figure 1.

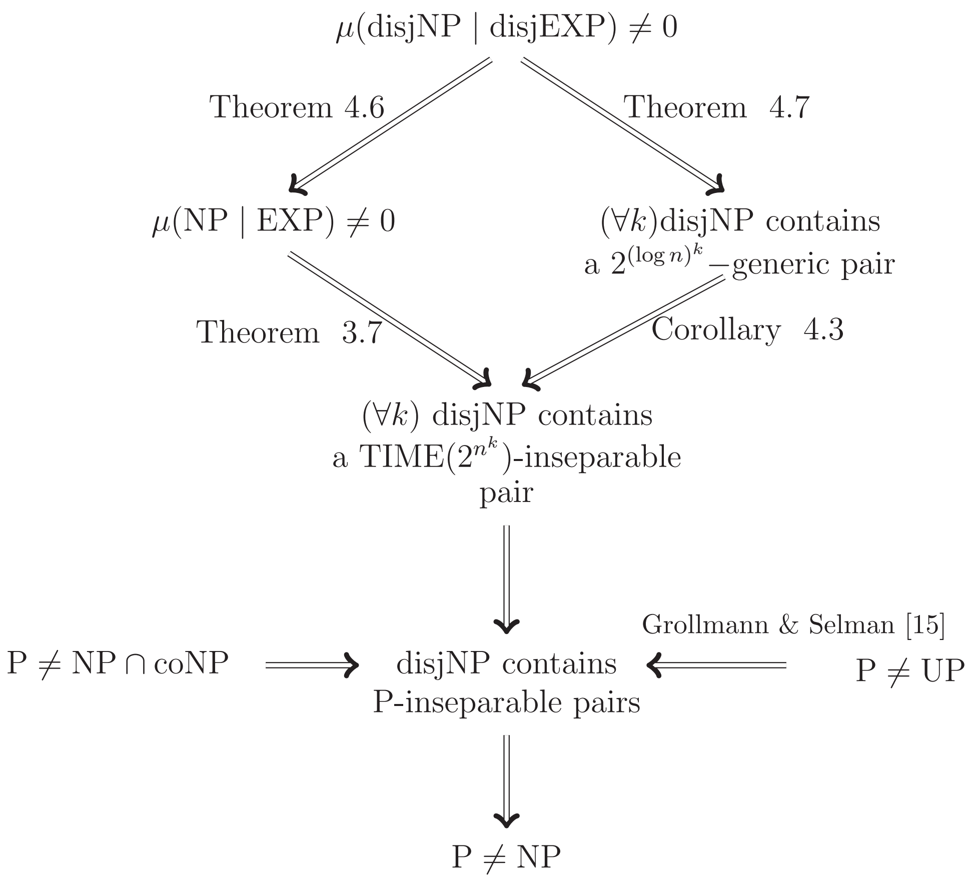

μ(disjNP | disjEXP) ≠ 0
Theorem 4.6
Theorem 4.7
μ(NP | EXP) ≠ 0
(∀k)disjNP contains a 2^((log n)^k)−generic pair
Theorem 3.7
Corollary 4.3
(∀k) disjNP contains a TIME(2^(n^k))-inseparable pair
Grollmann & Selman [15]
P ≠ NP ∩ coNP
disjNP contains P-inseparable pairs
P ≠ UP
P ≠ NP

Figure 1: Relations among some strong hypotheses.

Theorem 5.1 (Homer-Selman [17], Fortnow-Rogers [8])**.** *There exists oracles A and B such that*

- $\mathrm{P}^A \neq \mathrm{NP}^A$ *and* disjNP^A *does not contain* P^A*-inseparable pairs.*
- $\mathrm{P}^B = \mathrm{NP}^B \cap \mathrm{coNP}^B = \mathrm{UP}^B$ *and* disjNP^B *does contain* P^B*-inseparable pairs.*

Theorem 5.2. *There exists an oracle C such that* $\mathrm{P}^C \neq \mathrm{UP}^C$ *but* NP^C *is contained in* $\mathrm{TIME}^C(n^{O(\log n)})$*. In particular this means that relative to C,* disjNP *contains* P*-inseparable pairs but there is a k (and in fact any real $k > 0$) such that* disjNP *has no* $\mathrm{TIME}(2^{n^k})$*-inseparable pairs.*

Theorem 5.3. *There exists a relativized world D, relative to which for all k,* disjNP *contains a* $\mathrm{TIME}(2^{n^k})$*-inseparable pair but* $\mu(\mathrm{NP}|\mathrm{EXP}) = 0$ *and* disjNP *does not contain a* $2^{(\log n)^k}$*-generic pair.*

Theorem 5.4. *There exists an oracle E relative to which for all k,* disjNP *contains a* $2^{(\log n)^k}$*-generic pair but* $\mu(\mathrm{disjNP}|\mathrm{disjEXP}) = 0$.

Conjecture 5.5. There exists an oracle H relative to which $\mu(\mathrm{NP}|\mathrm{EXP}) \neq 0$ but $\mu(\mathrm{disjNP}|\mathrm{disjEXP}) = 0$.

Let K be a PSPACE-compete set, R be a "random" oracle and let

$$H = K \oplus R = \{\langle 0, x\rangle \mid x \in K\} \cup \{\langle 1, y\rangle \mid y \in R\}.$$

Kautz and Miltersen show in [20] that relative to H, $\mu(\mathrm{NP}|\mathrm{EXP}) \neq 0$. Kahn, Saks and Smyth [19] combined with unpublished work of Impagliazzo and Rudich show that relative to H there is a polynomial-time algorithm that solves languages in $\mathrm{NP} \cap \mathrm{coNP}$ on average for infinitely-many lengths which would imply $\mu(\mathrm{NP} \cap \mathrm{coNP}|\mathrm{EXP}) = 0$ relative to H. We conjecture that one can modify this proof to show $\mu(\mathrm{disjNP}^H|\mathrm{disjEXP}^H) = 0$.

References

[1] K. Ambos-Spies, H. Fleischhack, and H. Huwig. Diagonalizations over polynomial time computable sets. *Theoretical Computer Science*, 51:177–204, 1987.

[2] K. Ambos-Spies and E. Mayordomo. Resource-bounded measure and randomness. In A. Sorbi, editor, *Complexity, Logic and Recursion Theory*, Lecture Notes in Pure and Applied Mathematics, pages 1–47. Marcel Dekker, New York, N.Y., 1997.

[3] K. Ambos-Spies, S. A. Terwijn, and X. Zheng. Resource bounded randomness and weakly complete problems. *Theoretical Computer Science*, 172:195–207, 1997.

[4] T. Baker, J. Gill, and R. Solovay. Relativizations of the P =? NP question. *SIAM Journal on Computing*, 4:431–442, 1975.

[5] J.M. Breutzmann and J.H. Lutz. Equivalence of measures of complexity classes. *SIAM Journal on Computing*, 29(1):302–326, 1999.

[6] S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.

[7] S. Even, A. L. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.

[8] L. Fortnow and J. Rogers. Separability and one-way functions. *Computational Complexity*, 11:137–157, 2002.

[9] C. Glaßer, A. L. Selman, and S. Sengupta. Reductions between disjoint NP-pairs. *Information and Computation*, 200:247–267, 2005.

[10] C. Glaßer, A. L. Selman, S. Sengupta, and L. Zhang. Disjoint NP-pairs. *SIAM Journal on Computing*, 33:1369–1416, 2004.

[11] C. Glaßer, A. L. Selman, and L. Zhang. Survey of disjoint NP-pairs and relations to propositional proof systems. In *Theoretical Computer Science: Essays in Memory of Shimon Even*, pages 241–253. Springer, 2006.

[12] C. Glaßer, A. L. Selman, and L. Zhang. Canonical disjoint NP-pairs of propositional proof systems. *Theoretical Computer Science*, 370:60–73, 2007.

[13] C. Glaßer, A. L. Selman, and L. Zhang. The informational content of canonical disjoint NP-pairs. In *COCOON*, LNCS, pages 307–317. Springer, 2007.

[14] O. Goldreich. On promise problems: A survey. In *Theoretical Computer Science: Essays in Memory of Shimon Even*, pages 254–290. Springer, 2006.

[15] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 11:309–335, 1988.

[16] J.M. Hitchcock. Resource-bounded measure bibliography. http://www.cs.uwyo.edu/˜jhitchco/bib/rbm.shtml.

[17] S. Homer and A. L. Selman. Oracles for structural properties: The isomorphism problem and public-key cryptography. *Journal of Computer and System Sciences*, 44:287–301, 1992.

[18] D.W. Juedes and J.H. Lutz. Weak completeness in E and E_2, 1995.

[19] J. Kahn, M. E. Saks, and C. D. Smyth:. A dual version of reimer's inequality and a proof of rudich's conjecture. In *Proceedings of the 15th Annual IEEE Conference on Computational Complexity*, pages 98–103, 2000.

[20] S. M. Kautz and P. B. Miltersen. Relative to a random oracle, NP is not small. *Journal of Computer and System Sciences*, 53:235–250, 1996.

[21] A.K. Lorentz and J.H. Lutz. Genericity and randomness over feasible probability measures. *Theoretical Computer Science*, 207(1):245–259, 1998.

[22] J. H. Lutz. Almost everywhere high nonuniform complexity. *Journal of Computer and System Sciences*, 44(2):220–258, 1992.

[23] J. H. Lutz. The quantitative structure of exponential time. In L. A. Hemaspaandra and A. L. Selman, editors, *Complexity Theory Retrospective II*, pages 225–254. Springer-Verlag, 1997.

[24] J. H. Lutz and E. Mayordomo. Twelve problems in resource-bounded measure. In G. Păun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science, entering the 21st century*, pages 83–101. World Scientific Publishing, 2001.

[25] A. Razborov. On provably disjoint NP pairs. Technical Report 94-006, ECCC, 1994.

[26] A.L. Selman. Complexity issues in cryptography. In *Computational complexity theory (Atlanta, GA, 1988)*, volume 38 of *Proc. Sympos. Appl. Math.*, pages 92–107. Amer. Math. Soc., 1989.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 405-416
www.stacs-conf.org

BRANCHING-TIME MODEL CHECKING OF ONE-COUNTER PROCESSES

STEFAN GÖLLER [1] AND MARKUS LOHREY [2]

[1] Universität Bremen, Fachbereich Mathematik und Informatik
E-mail address: goeller@informatik.uni-bremen.de

[2] Universität Leipzig, Institut für Informatik
E-mail address: lohrey@informatik.uni-leipzig.de

ABSTRACT. One-counter processes (OCPs) are pushdown processes which operate only on a unary stack alphabet. We study the computational complexity of model checking computation tree logic (CTL) over OCPs. A PSPACE upper bound is inherited from the modal μ-calculus for this problem. First, we analyze the periodic behaviour of CTL over OCPs and derive a model checking algorithm whose running time is exponential only in the number of control locations and a syntactic notion of the formula that we call leftward until depth. Thus, model checking fixed OCPs against CTL formulas with a fixed leftward until depth is in P. This generalizes a result of the first author, Mayr, and To for the expression complexity of CTL's fragment EF. Second, we prove that already over some fixed OCP, CTL model checking is PSPACE-hard. Third, we show that there already exists a fixed CTL formula for which model checking of OCPs is PSPACE-hard. For the latter, we employ two results from complexity theory: (i) Converting a natural number in Chinese remainder presentation into binary presentation is in logspace-uniform NC^1 and (ii) PSPACE is AC^0-serializable. We demonstrate that our approach can be used to answer further open questions.

1. Introduction

Pushdown automata (PDAs) (or recursive state machines) are a natural model for sequential programs with recursive procedure calls, and their verification problems have been studied extensively. The complexity of model checking problems for PDAs is quite well understood: The reachability problem for PDAs can be solved in polynomial time [4, 10]. Model checking modal μ-calculus over PDAs was shown to be EXPTIME-complete in [29], and the global version of the model checking problem has been considered in [7, 21, 22]. The EXPTIME lower bound for model checking PDAs also holds for the simpler logic CTL and its fragment EG [28], even for a fixed formula (data complexity) [5] or a fixed PDA (expression complexity). On the other hand, model checking PDAs against the logic EF (another natural fragment of CTL) is PSPACE-complete [28], and again the lower bound still holds if either the formula or the PDA is fixed [4]. Model checking

1998 ACM Subject Classification: F.4.1; F.1.3.
Key words and phrases: model checking, computation tree logic, complexity theory.
The second author would like to acknowledge the support by DFG research project GELO.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2472

problems for various fragments and extensions of PDL (Propositional Dynamic Logic) over PDAs were studied in [12].

One-counter processes (OCPs) are Minsky counter machines with just one counter. They can also be seen as a special case of PDAs with just one stack symbol, plus a non-removable bottom symbol which indicates an empty stack (and thus allows to test the counter for zero) and hence constitute a natural and fundamental computational model. In recent years, model checking problems for OCPs received increasing attention [13, 15, 23, 25]. Clearly, all upper complexity bounds carry over from PDAs. The question, whether these upper bounds can be matched by lower bounds was just recently solved for several important logics: Model checking modal μ-calculus over OCPs is PSPACE-complete. The PSPACE upper bound was shown in [23], and a matching lower bound can easily be shown by a reduction from emptiness of alternating unary finite automata, which was shown to be PSPACE-complete in [18, 19]. This lower bound even holds if either the OCP or the formula is fixed. The situation becomes different for the fragment EF. In [13], it was shown that model checking EF over OCPs is in the complexity class P^{NP} (the class of all problems that can be solved on a deterministic polynomial time machine with access to an oracle from NP). Moreover, if the input formula is represented succinctly as a directed acyclic graph, then model checking EF over OCPs is also hard for P^{NP}. For the standard (and less succinct) tree representation for formulas, only hardness for the class $\mathsf{P}^{\mathsf{NP}[\log]}$ (the class of all problems that can be solved on a deterministic polynomial time machine which is allowed to make $O(\log(n))$ many queries to an oracle from NP) was shown in [13]. In fact, there already exists a fixed EF formula such that model checking this formula over a given OCP is hard for $\mathsf{P}^{\mathsf{NP}[\log]}$, i.e., the data complexity is $\mathsf{P}^{\mathsf{NP}[\log]}$-hard.

In this paper we consider the model checking problem for CTL over OCPs. By the known upper bound for the modal μ-calculus [23] this problem belongs to PSPACE. First, we analyze the combinatorics of CTL model checking over OCPs. More precisely, we analyze the periodic behaviour of the set of natural numbers that satisfy a given CTL formula in a given control location of the OCP (Thm. 4.1). By making use of Thm. 4.1, we can derive a model checking algorithm whose running time is exponential only in the number of control locations and a syntactic measure on CTL formulas that we call leftward until depth (Thm. 4.2). As a corollary, we obtain that model checking a fixed OCP against CTL formulas of fixed leftward until depth lies in P. This generalizes a recent result from [13], where it was shown that the expression complexity of EF over OCPs lies in P. Next, we focus on lower bounds. We show that model checking CTL over OCPs is PSPACE-complete, even if we fix either the OCP (Thm. 5.3) or the CTL formula (Thm. 7.2). The proof of Thm. 5.3 uses a reduction from QBF. We have to construct a fixed OCP for which we can construct for a given unary encoded number i CTL formulas that express, when interpreted over our fixed OCP, whether the current counter value is divisible by 2^i and whether the i^{th} bit in the binary representation of the current counter value is 1, respectively. For the proof of Thm. 7.2 (PSPACE-hardness of data complexity for CTL) we use two techniques from complexity theory, which to our knowledge have not been applied in the context of verification so far: (i) the existence of small depth circuits for converting a number from Chinese remainder representation to binary representation and (ii) the fact that PSPACE-computations are serializable in a certain sense (see Sec. 6 for details). One of the main obstructions in getting lower bounds for OCPs is the fact that OCPs are well suited for testing divisibility properties of the counter value and hence can deal with numbers in Chinese remainder representation, but it is not clear how to deal with numbers in binary representation. Small depth circuits for converting a number from Chinese remainder representation to binary representation are the key in order to overcome this obstruction.

We are confident that our new lower bound techniques described above can be used for proving further lower bounds for OCPs. We present two other applications of our techniques in Sec. 8:

(i) We show that model checking EF over OCPs is complete for P^{NP} even if the input formula is represented by a tree (Thm. 8.1) and thereby solve an open problem from [13]. (ii) We improve a lower bound on a decision problem for one-counter Markov decision processes from [6] (Thm. 8.2). The following table summarizes the picture on the complexity of model checking for PDAs and OCPs. Our new results are marked with (*).

Logic	**PDA**	**OCP**
modal μ-calculus	EXPTIME-complete	PSPACE-complete
modal μ-calculus, fixed formula	EXPTIME-complete	PSPACE-complete
modal μ-calculus, fixed system	EXPTIME-complete	PSPACE-complete
CTL, fixed formula	EXPTIME-complete	PSPACE-complete (*)
CTL, fixed system	EXPTIME-complete	PSPACE-complete (*)
CTL, fixed system, fixed leftward until depth	EXPTIME-complete	in P (*)
EF	PSPACE-complete	P^{NP}-complete (*)
EF, fixed formula	PSPACE-complete	$\mathsf{P}^{\mathsf{NP}[\log]}$-hard, in P^{NP}
EF, fixed system	PSPACE-complete	in P

Missing proofs due to space restrictions can be found in the full version of this paper [14].

2. Preliminaries

We denote the naturals by $\mathbb{N} = \{0, 1, 2, \ldots\}$. For $i, j \in \mathbb{N}$ let $[i, j] = \{k \in \mathbb{N} \mid i \leq k \leq j\}$ and $[j] = [1, j]$. In particular $[0] = \emptyset$. For $n \in \mathbb{N}$ and $i \geq 1$, let $\text{bit}_i(n)$ denote the i^{th} least significant bit of the binary representation of n, i.e., $n = \sum_{i \geq 1} 2^{i-1} \cdot \text{bit}_i(n)$. For every finite and non-empty subset $M \subseteq \mathbb{N} \setminus \{0\}$, define $\text{LCM}(M)$ to be the *least common multiple* of all numbers in M. It is known that $2^k \leq \text{LCM}([k]) \leq 4^k$ for all $k \geq 9$ [20]. As usual, for a possibly infinite alphabet A, A^* (resp. A^ω) denotes the set of all finite (resp. infinite) words over A. Let $A^\infty = A^* \cup A^\omega$ and $A^+ = A^* \setminus \{\varepsilon\}$, where ε is the empty word. The length of a finite word w is denoted by $|w|$. For a word $w = a_1 a_2 \cdots a_n \in A^*$ (resp. $w = a_1 a_2 \cdots \in A^\omega$) with $a_i \in A$ and $i \in [n]$ (resp. $i \geq 1$), we denote by w_i the i^{th} letter a_i. A nondeterministic finite automaton (NFA) is a tuple $A = (S, \Sigma, \delta, s_0, S_f)$, where S is a finite set of *states*, Σ is a *finite alphabet*, $\delta \subseteq S \times \Sigma \times S$ is the *transition relation*, $s_0 \in S$ is the *initial state*, and $S_f \subseteq S$ is a set of *final states*. We assume some basic knowledge in complexity theory, see e.g. [1] for more details.

3. One-counter processes and computation tree logic

Fix a countable set $\mathcal{P}$ of *propositions*. A *transition system* is a triple $T = (S, \{S_p \mid p \in \mathcal{P}\}, \rightarrow)$, where S is the set of *states*, $\rightarrow \subseteq S \times S$ is the set of *transitions* and $S_p \subseteq S$ for all $p \in \mathcal{P}$ with $S_p = \emptyset$ for all but finitely many $p \in \mathcal{P}$. We write $s_1 \rightarrow s_2$ instead of $(s_1, s_2) \in \rightarrow$. The set of all *finite* (resp. *infinite*) *paths* in T is $\text{path}_+(T) = \{\pi \in S^+ \mid \forall i \in [|\pi| - 1] : \pi_i \rightarrow \pi_{i+1}\}$ (resp. $\text{path}_\omega(T) = \{\pi \in S^\omega \mid \forall i \geq 1 : \pi_i \rightarrow \pi_{i+1}\}$). For a subset $U \subseteq S$ of states, a (finite or infinite) path π is called a *U-path* if $\pi \in U^\infty$.

A *one-counter process* (OCP) is a tuple $\mathbb{O} = (Q, \{Q_p \mid p \in \mathcal{P}\}, \delta_0, \delta_{>0})$, where Q is a finite set of *control locations*, $Q_p \subseteq Q$ for all $p \in \mathcal{P}$ with $Q_p = \emptyset$ for all but finitely many $p \in \mathcal{P}$, $\delta_0 \subseteq Q \times \{0, 1\} \times Q$ is a set of *zero transitions*, and $\delta_{>0} \subseteq Q \times \{-1, 0, 1\} \times Q$ is a set of *positive transitions*. The *size* of the OCP $\mathbb{O}$ is $|\mathbb{O}| = |Q| + \sum_{p \in \mathcal{P}} |Q_p| + |\delta_0| + |\delta_{>0}|$. The transition system defined by $\mathbb{O}$ is $T(\mathbb{O}) = (Q \times \mathbb{N}, \{Q_p \times \mathbb{N} \mid p \in \mathcal{P}\}, \rightarrow)$, where $(q, n) \rightarrow (q', n + k)$ if and only if either $n = 0$ and $(q, k, q') \in \delta_0$, or $n > 0$ and $(q, k, q') \in \delta_{>0}$. A *one-counter net* (OCN) is an OCP, where $\delta_0 \subseteq \delta_{>0}$. For $(q, k, q') \in \delta_0 \cup \delta_{>0}$ we usually write $q \xrightarrow{k} q'$.

More details on the temporal logic CTL can be found for instance in [2]. *Formulas* φ of CTL are defined by the following grammar, where $p \in \mathcal{P}$:

$$\varphi \quad ::= \quad p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists\mathsf{X}\varphi \mid \exists\varphi\mathsf{U}\varphi \mid \exists\varphi\mathsf{WU}\varphi.$$

Given a transition system $T = (S, \{S_p \mid p \in \mathcal{P}\}, \rightarrow)$ and a CTL formula φ, we define the semantics $[\![\varphi]\!]_T \subseteq S$ by induction on the structure of φ as follows: $[\![p]\!]_T = S_p$ for each $p \in \mathcal{P}$, $[\![\neg\varphi]\!]_T = S \setminus [\![\varphi]\!]_T$, $[\![\varphi_1 \wedge \varphi_2]\!]_T = [\![\varphi_1]\!]_T \cap [\![\varphi_2]\!]_T$, $[\![\exists\mathsf{X}\varphi]\!]_T = \{s \in S \mid \exists s' \in [\![\varphi]\!]_T : s \rightarrow s'\}$, $[\![\exists\varphi_1\mathsf{U}\varphi_2]\!]_T = \{s \in S \mid \exists\pi \in \mathrm{path}_+(T) : \pi_1 = s, \pi_{|\pi|} \in [\![\varphi_2]\!]_T, \forall i \in [|\pi| - 1] : \pi_i \in [\![\varphi_1]\!]_T\}$, $[\![\exists\varphi_1\mathsf{WU}\varphi_2]\!]_T = [\![\exists\varphi_1\mathsf{U}\varphi_2]\!]_T \cup \{s \in S \mid \exists\pi \in \mathrm{path}_\omega(T) : \pi_1 = s, \forall i \geq 1 : \pi_i \in [\![\varphi_1]\!]_T\}$. We also write $(T, s) \models \varphi$ (or briefly $s \models \varphi$ if T is clear from the context) for $s \in [\![\varphi]\!]_T$. We introduce the usual abbreviations $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\forall\mathsf{X}\varphi = \neg\exists\mathsf{X}\neg\varphi$, $\exists\mathsf{F}\varphi = \exists(p \vee \neg p)\mathsf{U}\varphi$, and $\exists\mathsf{G}\varphi = \exists\varphi\mathsf{WU}(p \wedge \neg p)$ for some $p \in \mathcal{P}$. Formulas of the CTL-fragment EF are given by the following grammar, where $p \in \mathcal{P}$: $\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists\mathsf{X}\varphi \mid \exists\mathsf{F}\varphi$. The *size* of CTL formulas is defined as follows: $|p| = 1$, $|\neg\varphi| = |\exists\mathsf{X}\varphi| = |\varphi| + 1$, $|\varphi_1 \wedge \varphi_2| = |\varphi_1| + |\varphi_2| + 1$, $|\exists\varphi_1\mathsf{U}\varphi_2| = |\exists\varphi_1\mathsf{WU}\varphi_2| = |\varphi_1| + |\varphi_2| + 1$.

4. CTL on OCPs: Periodic behaviour and upper bounds

The goal of this section is to prove a periodicity property of CTL over OCPs, which implies an upper bound for CTL on OCPs, see Thm. 4.2. As a corollary, we state that for a fixed OCP, CTL model checking restricted to formulas of fixed leftward until depth (see the definition below) can be done in polynomial time. We define the *leftward until depth* lud of CTL formulas inductively as follows: $\mathrm{lud}(p) = 0$ for $p \in \mathcal{P}$, $\mathrm{lud}(\neg\varphi) = \mathrm{lud}(\exists\mathsf{X}\varphi) = \mathrm{lud}(\varphi)$, $\mathrm{lud}(\varphi_1 \wedge \varphi_2) = \max\{\mathrm{lud}(\varphi_1), \mathrm{lud}(\varphi_2)\}$, $\mathrm{lud}(\exists\varphi_1\mathsf{U}\varphi_2) = \mathrm{lud}(\exists\varphi_1\mathsf{WU}\varphi_2) = \max\{\mathrm{lud}(\varphi_1) + 1, \mathrm{lud}(\varphi_2)\}$. A similar definition of until depth can be found in [24], but there the until depth of $\exists\varphi_1\mathsf{U}\varphi_2$ is 1 plus the maximum of the until depths of φ_1 and φ_2. Note that $\mathrm{lud}(\varphi) \leq 1$ for every EF formula φ.

Let us fix an OCP $\mathbb{O} = (Q, \{Q_p \mid p \in \mathcal{P}\}, \delta_0, \delta_{>0})$ for the rest of this section. Let $|Q| = k$ and define $K = \mathrm{LCM}([k])$ and $K_\varphi = K^{\mathrm{lud}(\varphi)}$ for each CTL formula φ.

Theorem 4.1. *For all* CTL *formulas* φ*, all* $q \in Q$ *and all* $n, n' > 2 \cdot |\varphi| \cdot k^2 \cdot K_\varphi$ *with* $n \equiv n'$ *mod* K_φ*:*

$$(q, n) \in [\![\varphi]\!]_{T(\mathbb{O})} \iff (q, n') \in [\![\varphi]\!]_{T(\mathbb{O})}. \tag{4.1}$$

Proof sketch. We prove the theorem by induction on the structure of φ. We only treat the difficult case $\varphi = \exists\psi_1\mathsf{U}\psi_2$ here. Let $T = \max\{2 \cdot |\psi_i| \cdot k^2 \cdot K_{\psi_i} \mid i \in \{1, 2\}\}$. Let us prove equivalence (4.1). Note that $K_\varphi = \mathrm{LCM}\{K \cdot K_{\psi_1}, K_{\psi_2}\}$ by definition. Let us fix an arbitrary control location $q \in Q$ and naturals $n, n' \in \mathbb{N}$ such that $2 \cdot |\varphi| \cdot k^2 \cdot K_\varphi < n < n'$ and $n \equiv n'$ mod K_φ. We have to prove that $(q, n) \in [\![\varphi]\!]_{T(\mathbb{O})}$ if and only if $(q, n') \in [\![\varphi]\!]_{T(\mathbb{O})}$. For this, let $d = n' - n$, which is a multiple of K_φ. We only treat the "if"-direction here and recommend the reader to consult [14] for helpful illustrations. So let us assume that $(q, n') \in [\![\varphi]\!]_{T(\mathbb{O})}$. To prove that $(q, n) \in [\![\varphi]\!]_{T(\mathbb{O})}$, we will use the following claim.

Claim: Assume some $[\![\psi_1]\!]_{T(\mathbb{O})}$-path $\pi = [(q_1, n_1) \rightarrow (q_2, n_2) \rightarrow \cdots \rightarrow (q_l, n_l)]$ with $n_i > T$ for all $i \in [l]$ and $n_1 - n_l \geq k^2 \cdot K \cdot K_{\psi_1}$. Then there exists a $[\![\psi_1]\!]_{T(\mathbb{O})}$-path from (q_1, n_1) to $(q_l, n_l + K \cdot K_{\psi_1})$, whose counter values are all strictly above $T + K \cdot K_{\psi_1}$.

The claim tells us that paths that lose height at least $k^2 \cdot K \cdot K_{\psi_1}$ and whose states all have counter values strictly above T can be flattened (without changing the starting state) by height $K \cdot K_{\psi_1}$.

Proof of the claim. For each counter value $h \in \{n_i \mid i \in [l]\}$ that appears in π, let $\mu(h) = \min\{i \in [l] \mid n_i = h\}$ denote the minimal position in π whose corresponding state has counter value h. Define $\Delta = k \cdot K_{\psi_1}$. We will be interested in $k \cdot K$ many consecutive intervals (of counter values) each of size Δ. Define the bottom $b = n_1 - (k \cdot K) \cdot \Delta$. Formally, an *interval* is a set $I_i = [b + (i-1) \cdot \Delta, b + i \cdot \Delta]$ for some $i \in [k \cdot K]$. Since each interval has size $\Delta = k \cdot K_{\psi_1}$, we can think of each interval I_i to consist of k consecutive *sub-intervals* of size K_{ψ_1} each. Note that each sub-interval has two extremal elements, namely its *upper* and *lower boundary*. Thus all k sub-intervals have $k + 1$ boundaries in total. Hence, by the pigeonhole principle, for each interval I_i, there exists some $c_i \in [k]$ and two distinct boundaries $\beta(i, 1) > \beta(i, 2)$ of distance $c_i \cdot K_{\psi_1}$ such that the control location of π's earliest state of counter value $\beta(i, 1)$ agrees with the control location of π's earliest state of counter value $\beta(i, 2)$, i.e., formally $q_{\mu(\beta(i,1))} = q_{\mu(\beta(i,2))}$. Observe that flattening the path π by gluing together π's states at position $\mu(\beta(i, 1))$ and $\mu(\beta(i, 2))$ (for this, we add $c_i \cdot K_{\psi_1}$ to each counter value at a position $\geq \beta(i, 2)$) still results in a $[\![\psi_1]\!]_{T(\mathbb{O})}$-path by induction hypothesis, since we reduced the height of π by a multiple of K_{ψ_1}. Our overall goal is to flatten π by gluing together states only of certain intervals such that we obtain a path whose height is in total by precisely $K \cdot K_{\psi_1}$ smaller than π's. Recall that there are $k \cdot K$ many intervals. By the pigeonhole principle there is some $c \in [k]$ such that $c_i = c$ for at least K many intervals I_i. By gluing together $\frac{K}{c} \in \mathbb{N}$ pairs of states of distance $c \cdot K_{\psi_1}$ each, we reduce π's height by exactly $\frac{K}{c} \cdot c \cdot K_{\psi_1} = K \cdot K_{\psi_1}$. This proves the claim.

Let us finish the proof the "if"-direction. Since by assumption $(q, n') \in [\![\varphi]\!]_{T(\mathbb{O})}$, there exists a finite path $\pi = (q_1, n_1) \to (q_2, n_2) \to \cdots \to (q_l, n_l)$, where $\pi[1, l-1]$ is a $[\![\psi_1]\!]_{T(\mathbb{O})}$-path, $(q, n') = (q_1, n_1)$, and where $(q_l, n_l) \in [\![\psi_2]\!]_{T(\mathbb{O})}$. To prove $(q, n) \in [\![\varphi]\!]_{T(\mathbb{O})}$, we will assume that $n_j > T$ for each $j \in [l]$. The case when $n_j = T$ for some $j \in [l]$ can be proven similarly. Assume first that the path $\pi[1, l-1]$ contains two states whose counter difference is at least $k^2 \cdot K \cdot K_{\psi_1} + K_\varphi$ which is (strictly) greater than $k^2 \cdot K \cdot K_{\psi_1}$. Since K_φ is a multiple of $K \cdot K_{\psi_1}$ by definition, we can apply the above claim $\frac{K_\varphi}{K \cdot K_{\psi_1}} \in \mathbb{N}$ many times to $\pi[1, l-1]$. This reduces the height by K_φ. We repeat this flattening process of $\pi[1, l-1]$ by height K_φ as long as possible, i.e., until any two states have counter difference smaller than $k^2 \cdot K \cdot K_{\psi_1} + K_\varphi$. Let σ denote the $[\![\psi_1]\!]_{T(\mathbb{O})}$-path starting in (q, n') that we obtain from $\pi[1, l-1]$ by this process. Thus, σ ends in some state, whose counter value is congruent n_{l-1} modulo K_φ (since we flattened $\pi[1, l-1]$ by a multiple of K_φ). Since K_φ is in turn a multiple of K_{ψ_2}, we can build a path σ' which extends the path σ by a single transition to some state that satisfies ψ_2 by induction hypothesis. Moreover, by our flattening process, the counter difference between any two states in σ' is at most $k^2 \cdot K \cdot K_{\psi_1} + K_\varphi \leq 2 \cdot k^2 \cdot K_\varphi$. Recall that $T = \max\{2 \cdot |\psi_i| \cdot k^2 \cdot K_{\psi_i} \mid i \in \{1, 2\}\}$. As

$$n \; > \; 2 \cdot |\varphi| \cdot k^2 \cdot K_\varphi \; = \; 2 \cdot (|\varphi| - 1 + 1) \cdot k^2 \cdot K_\varphi \; \geq \; T + 2 \cdot k^2 \cdot K_\varphi,$$

it follows that the path that results from σ' by subtracting d from each counter value (this path starts in (q, n)) is strictly above T. Moreover, since d is a multiple of K_{ψ_1} and K_{ψ_2}, this path witnesses $(q, n) \in [\![\varphi]\!]_{T(\mathbb{O})}$ by induction hypothesis. ∎

The following result can be obtained basically by using the standard model checking algorithm for CTL on finite systems (see e.g. [2]) in combination with Thm. 4.1.

Theorem 4.2. *For a given one-counter process* $\mathbb{O} = (Q, \{Q_p \mid p \in \mathcal{P}\}, \delta_0, \delta_{>0})$, *a* CTL *formula* φ, *a control location* $q \in Q$, *and* $n \in \mathbb{N}$ *given in binary, one can decide* $(q, n) \in [\![\varphi]\!]_{T(\mathbb{O})}$ *in time* $O(\log(n) + |Q|^3 \cdot |\varphi|^2 \cdot 4^{|Q| \cdot \mathrm{lud}(\varphi)} \cdot |\delta_0 \cup \delta_{>0}|)$.

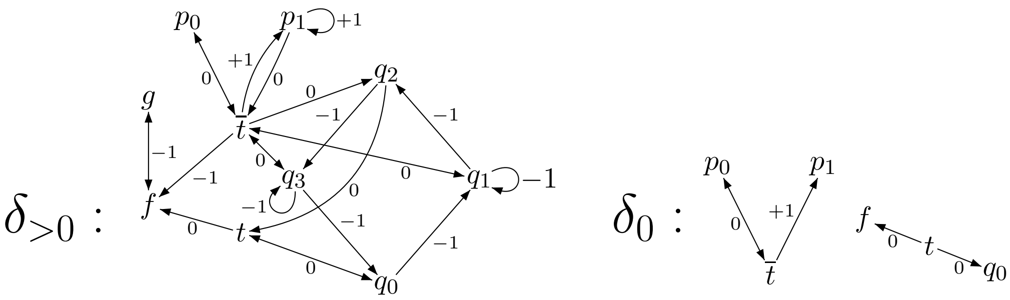


Figure 1: The one-counter net $\mathbb{O}$ for which CTL model checking is PSPACE-hard

As a corollary, we can deduce that for every fixed OCP $\mathbb{O}$ and every fixed k the question if for a given state s and a given CTL formula φ with $\mathrm{lud}(\varphi) \leq k$, we have $(T(\mathbb{O}), s) \models \varphi$, is in P. This generalizes a result from [13], stating that the expression complexity of EF over OCPs is in P.

5. Expression complexity for CTL is hard for PSPACE

The goal of this section is to prove that model checking CTL is PSPACE-hard already over a fixed OCN. We show this via a reduction from the well-known PSPACE-complete problem QBF. Our lower bound proof is separated into three steps. In step one, we define a family of CTL formulas $(\varphi_i)_{i\geq 1}$ such that over the fixed OCN $\mathbb{O}$ that is depicted in Fig. 1 we can express (non-)divisibility by 2^i. In step two, we define a family of CTL formulas $(\psi_i)_{i\geq 1}$ such that over $\mathbb{O}$ we can express if the i^{th} bit in the binary representation of a natural is set to 1. In our final step, we give the reduction from QBF. For step one, we need the following simple fact which characterizes divisibility by powers of two (recall that $[n] = \{1, \dots, n\}$, in particular $[0] = \emptyset$):

$$\forall n \geq 0, i \geq 1 : \ 2^i \text{ divides } n \ \Leftrightarrow \ (2^{i-1} \text{ divides } n \ \wedge \ |\{n' \in [n] \mid 2^{i-1} \text{ divides } n'\}| \text{ is even}) \quad (5.1)$$

The set of propositions of $\mathbb{O}$ in Fig. 1 coincides with its control locations. Recall that $\mathbb{O}$'s zero transitions are denoted by δ_0 and $\mathbb{O}$'s positive transitions are denoted by $\delta_{>0}$. Since $\delta_0 \subseteq \delta_{>0}$, $\mathbb{O}$ is indeed an OCN. Note that both t and $\overline{t}$ are control locations of $\mathbb{O}$. Now we define a family of CTL formulas $(\varphi_i)_{i\geq 1}$ such that for each $n \in \mathbb{N}$ we have: (i) $(t, n) \models \varphi_i$ if and only if 2^i divides n and (ii) $(\overline{t}, n) \models \varphi_i$ if and only if 2^i does *not* divide n. On first sight, it might seem superfluous to let the control location t represent divisibility by powers of two and the control location $\overline{t}$ to represent non-divisibility by powers of two since CTL allows negation. However the fact that we have *only one* family of formulas $(\varphi_i)_{i\geq 1}$ to express both divisibility and non-divisibility is a crucial technical subtlety that is necessary in order to avoid an exponential blowup in formula size. By making use of (5.1), we construct the formulas φ_i inductively. First, let us define the auxiliary formulas $\text{test} = t \vee \overline{t}$ and $\varphi_\diamond = q_0 \vee q_1 \vee q_2 \vee q_3$. Think of $\varphi_\diamond$ to hold in those control locations that altogether are situated in the "diamond" in Fig. 1. We define

$$\begin{aligned} \varphi_1 &= \text{test} \wedge \exists \mathsf{X}\ (f \wedge \mathsf{EF}(f \wedge \neg\exists\mathsf{X} g)) \text{ and} \\ \varphi_i &= \text{test} \ \wedge \ \exists\mathsf{X}\left(\exists(\varphi_\diamond \wedge \exists\mathsf{X}\varphi_{i-1}) \ \mathsf{U}\ (q_0 \wedge \neg\exists\mathsf{X} q_1)\right) \text{ for } i > 1. \end{aligned}$$

Since φ_{i-1} is only used once in φ_i, we get $|\varphi_i| \in O(i)$. The following lemma states the correctness of the construction.

Lemma 5.1. *Let $n \geq 0$ and $i \geq 1$. Then*

- $(t, n) \models \varphi_i$ *if and only if* 2^i *divides* n.

- $(\overline{t}, n) \models \varphi_i$ *if and only if* 2^i *does not divide* n.

Proof sketch. The lemma is proved by induction on i. The induction base for $i = 1$ is easy to check. For $i > 1$, observe that φ_i can only be true either in control location t or $\overline{t}$. Note that the formula right to the until symbol in φ_i expresses that we are in q_0 and that the current counter value is zero. Also note that the formula left to the until symbol requires that $\varphi_\diamond$ holds, i.e., we are always in one of the four "diamond control locations". In other words, we decrement the counter by moving along the diamond control locations (by possibly looping at q_1 and q_3) and always check if $\exists\mathsf{X}\varphi_{i-1}$ holds, just until we are in q_0 and the counter value is zero. Since there are transitions from q_1 and q_3 to $\overline{t}$ (but not to t), the induction hypothesis implies that the formula $\exists\mathsf{X}\varphi_{i-1}$ can be only true in q_1 and q_3 as long as the current counter value is not divisible by 2^{i-1}. Similarly, since there are transitions from q_0 and q_2 to t (but not to $\overline{t}$), the induction hypothesis implies that the formula $\exists\mathsf{X}\varphi_{i-1}$ can be only true in q_0 and q_2 if the current counter value is divisible by 2^{i-1}. With (5.1) this implies the lemma. ■

For expressing if the i^{th} bit of a natural is set to 1, we make use of the following simple fact:

$$\forall n \geq 0, i \geq 1 : \text{bit}_i(n) = 1 \iff |\{n' \in [n] \mid 2^{i-1} \text{ divides } n'\}| \text{ is odd} \tag{5.2}$$

Let us now define a family of CTL formulas $(\psi_i)_{i\geq 1}$ such that for each $n \in \mathbb{N}$ we have $\text{bit}_i(n) = 1$ if and only if $(\overline{t}, n) \models \psi_i$. We set $\psi_1 = \varphi_1$ and $\psi_i = \overline{t} \wedge \exists\mathsf{X}\left((q_1 \vee q_2) \wedge \mu_i\right)$, where $\mu_i = \exists(\varphi_\diamond \wedge \exists\mathsf{X}\varphi_{i-1}) \,\mathsf{U}\, (q_0 \wedge \neg\exists\mathsf{X} q_1)$ for each $i > 1$. Due to the construction of ψ_i and since $|\varphi_i| \in O(i)$, we obtain that $|\psi_i| \in O(i)$. The following lemma states the correctness of the construction.

Lemma 5.2. *Let* $n \geq 0$ *and let* $i \geq 1$. *Then* $(\overline{t}, n) \models \psi_i$ *if and only if* $\text{bit}_i(n) = 1$.

Let us sketch the final step of the reduction from QBF. For this, let us assume some quantified Boolean formula $\alpha = Q_k x_k\, Q_{k-1} x_{k-1} \cdots Q_1 x_1 : \beta(x_1, \ldots, x_k)$, where β is a Boolean formula over variables $\{x_1, \ldots, x_k\}$ and $Q_i \in \{\exists, \forall\}$ is a quantifier for each $i \in [k]$. Think of each truth assignment $\vartheta : \{x_1, \ldots, x_k\} \to \{0, 1\}$ to correspond to the natural number $n(\vartheta) \in [0, 2^k - 1]$, where $\text{bit}_i(n(\vartheta)) = 1$ if and only if $\vartheta(x_i) = 1$, for each $i \in [k]$. Let $\widehat{\beta}$ be the CTL formula that is obtained from β by replacing each occurrence of x_i by ψ_i, which corresponds to applying Lemma 5.2. It remains to describe how we deal with quantification. Think of this as to consecutively incrementing the counter from state $(\overline{t}, 0)$ as follows. First, setting the variable x_k to 1 will correspond to adding 2^{k-1} to the counter and getting to state $(\overline{t}, 2^{k-1})$. Setting x_k to 0 on the other hand will correspond to adding 0 to the counter and hence remaining in state $(\overline{t}, 0)$. Next, setting x_{k-1} to 1 corresponds to adding to the current counter value 2^{k-2}, whereas setting x_{k-1} to 0 corresponds to adding 0, as expected. These incrementation steps can be achieved using the formulas φ_i from Lemma 5.1. Finally, after setting variable x_1 either to 0 or 1, we verify if the CTL formula $\widehat{\beta}$ holds. Formally, let $\bigcirc_i = \wedge$ if $Q_i = \exists$ and $\bigcirc_i = \rightarrow$ if $Q_i = \forall$ for each $i \in [k]$ (recall that $Q_k, \ldots, Q_1$ are the quantifiers of our quantified Boolean formula α). Let $\theta_1 = Q_1\mathsf{X}\left((p_0 \vee p_1) \bigcirc_1 \exists\mathsf{X}\,\widehat{\beta}\right)$ and for $i \in [2, k]$:

$$\theta_i = Q_i\mathsf{X}\ \left((p_0 \vee p_1) \bigcirc_i \exists \left((p_0 \vee \exists\mathsf{X}\,(\overline{t} \wedge \varphi_{i-1}))\ \mathsf{U}\ (\overline{t} \wedge \neg\varphi_{i-1} \wedge \theta_{i-1})\right)\right).$$

Then, it can be show that α is valid if and only if $(\overline{t}, 0) \in [\![\theta_k]\!]_{T(\mathbb{O})}$.

Theorem 5.3. CTL *model checking of the fixed OCN* $\mathbb{O}$ *from Fig. 1 is* PSPACE*-hard.*

Note that the constructed CTL formula has leftward until depth that depends on the size of α. By Thm. 4.2 this cannot be avoided unless P = PSPACE. Observe that in order to express divisibility by powers of two, our CTL formulas $(\varphi_i)_{i\geq 0}$ have linearly growing leftward until depth.

6. Tools from complexity theory

For Sec. 7 and 8 we need some concepts from complexity theory. By $\mathsf{P}^{\mathsf{NP}[\log]}$ we denote the class of all problems that can be solved on a polynomially time bounded deterministic Turing machines which can have access to an NP-oracle only logarithmically many times, and by P^{NP} the corresponding class without the restriction to logarithmically many queries. Let us briefly recall the definition of the circuit complexity class NC^1, more details can be found in [26]. We consider Boolean circuits $C = C(x_1, \dots, x_n)$ built up from AND- and OR-gates. Each input gate is labeled with a variable x_i or a negated variable $\neg x_i$. The output gates are linearly ordered. Such a circuit computes a function $f_C : \{0,1\}^n \to \{0,1\}^m$, where m is the number of output gates, in the obvious way. The *fan-in of a circuit* is the maximal number of incoming wires of a gate in the circuit. The *depth of a circuit* is the number of gates along a longest path from an input gate to an output gate. A *logspace-uniform* NC^1*-circuit family* is a sequence $(C_n)_{n\geq 1}$ of Boolean circuits such that for some polynomial $p(n)$ and constant c: (i) C_n contains at most $p(n)$ many gates, (ii) the depth of C_n is at most $c \cdot \log(n)$, (iii) the fan-in of C_n is at most 2, (iv) for each m there is at most one circuit in $(C_n)_{n\geq 1}$ with exactly m input gates, and (v) there exists a logspace transducer that computes on input 1^n a representation (e.g. as a node-labeled graph) of the circuit C_n. Such a circuit family computes a partial mapping on $\{0,1\}^*$ in the obvious way (note that we do not require to have for every $n \geq 0$ a circuit with exactly n input gates in the family, therefore the computed mapping is in general only partially defined). In the literature on circuit complexity one can find more restrictive notions of uniformity, see e.g. [26], but logspace uniformity suffices for our purposes. In fact, polynomial time uniformity suffices for proving our lower bounds w.r.t. polynomial time reductions.

For $m \geq 1$ and $0 \leq M \leq 2^m - 1$ let $\mathrm{BIN}_m(M) = \mathrm{bit}_m(M) \cdots \mathrm{bit}_1(M) \in \{0,1\}^m$ denote the m-bit binary representation of M. Let p_i denote the i^{th} prime number. It is well-known that the i^{th} prime requires $O(\log(i))$ bits in its binary representation. For a number $0 \leq M < \prod_{i=1}^m p_i$ we define the *Chinese remainder representation* $\mathrm{CRR}_m(M)$ as the Boolean tuple $\mathrm{CRR}_m(M) = (x_{i,r})_{i\in[m],0\leq r<p_i}$ with $x_{i,r} = 1$ if $M \bmod p_i = r$ and $x_{i,r} = 0$ else. By the following theorem, one can transform a Chinese remainder representation very efficiently into binary representation.

Theorem 6.1 ([9]). *There is a logspace-uniform* NC^1*-circuit family* $(B_m((x_{i,r})_{i\in[m],0\leq r<p_i}))_{m\geq 1}$ *such that for every* $m \geq 1$, B_m *has* m *output gates and for every* $0 \leq M < \prod_{i=1}^m p_i$ *we have that* $B_m(\mathrm{CRR}_m(M)) = \mathrm{BIN}_m(M \textit{ mod } 2^m)$.

By [17], we could replace logspace-uniform NC^1-circuits in Thm. 6.1 even by DLOGTIME-uniform TC^0-circuits. The existence of a P-uniform NC^1-circuit family for converting from Chinese remainder representation to binary representation was already shown in [3]. Usually the Chinese remainder representation of M is the tuple $(r_i)_{i\in[m]}$, where $r_i = M \bmod p_i$. Since the primes p_i will be always given in unary notation, there is no essential difference between this representation and our Chinese remainder representation. The latter is more suitable for our purpose.

The following definition of NC^1-serializability is a variant of the more classical notion of serializability [8, 16], which fits our purpose better. A language L is NC^1*-serializable* if there exists an NFA A over the alphabet $\{0,1\}$, a polynomial $p(n)$, and a logspace-uniform NC^1-circuit family $(C_n)_{n\geq 0}$, where C_n has exactly $n+p(n)$ many inputs and one output, such that for every $x \in \{0,1\}^n$ we have $x \in L$ if and only if $C_n(x, 0^{p(n)}) \cdots C_n(x, 1^{p(n)}) \in L(A)$, where "$\cdots$" refers to the lexicographic order on $\{0,1\}^{p(n)}$. With this definition, it can be shown that all languages in PSPACE are NC^1-serializable. A proof can be found in the appendix of [14]; it is just a slight adaptation of the proofs from [8, 16].

7. Data complexity for CTL is hard for PSPACE

In this section, we prove that also the data complexity of CTL over OCNs is hard for PSPACE and therefore PSPACE-complete by the known upper bounds for the modal μ-calculus [23]. Let us fix the set of propositions $\mathcal{P} = \{\alpha, \beta, \gamma\}$ for this section. In the following, w.l.o.g. we allow in δ_0 (resp. in $\delta_{>0}$) transitions of the kind (q, k, q'), where $k \in \mathbb{N}$ (resp. $k \in \mathbb{Z}$) is given in unary representation with the expected intuitive meaning.

Proposition 7.1. *For the fixed* EF *formula* $\varphi = (\alpha \to \exists\mathsf{X}(\beta \wedge \mathsf{EF}(\neg\exists\mathsf{X}\gamma)))$ *the following problem can be solved with a logspace transducer:*
INPUT: A list $p_1, \dots, p_m$ *of the first* m *consecutive (unary encoded) prime numbers and a Boolean formula* $F = F((x_{i,r})_{i\in[m],0\le r<p_i})$
OUTPUT: An OCN $\mathbb{O}(F)$ *with distinguished control locations* in *and* out, *such that for every number* $0 \le M < \prod_{i=1}^m p_i$ *we have that* $F(\mathrm{CRR}_m(M)) = 1$ *if and only if there exists a* $[\![\varphi]\!]_{T(\mathbb{O}(F))}$*-path from* (in, M) *to* (out, M) *in the transition system* $T(\mathbb{O}(F))$.

Proof. W.l.o.g., negations occur in F only in front of variables. Then additionally, a negated variable $\neg x_{i,r}$ can be replaced by the disjunction $\bigvee\{x_{i,k} \mid 0 \le k < p_i, r \ne k\}$. This can be done in logspace, since the primes p_i are given in unary. Thus, we can assume that F does not contain negations.

The idea is to traverse the Boolean formula F with the OCN $\mathbb{O}(F)$ in a depth first manner. Each time a variable $x_{i,r}$ is seen, the OCN may also enter another branch, where it is checked, whether the current counter value is congruent r modulo p_i. Let $\mathbb{O}(F) = (Q, \{Q_\alpha, Q_\beta, Q_\gamma\}, \delta_0, \delta_{>0})$, where $Q = \{\mathsf{in}(G), \mathsf{out}(G) \mid G \text{ is a subformula of } F\} \cup \{\mathrm{div}(p_1), \dots, \mathrm{div}(p_m), \bot\}$, $Q_\alpha = \{\mathsf{in}(x_{i,r}) \mid i \in [m], 0 \le r < p_i\}$, $Q_\beta = \{\mathrm{div}(p_1), \dots, \mathrm{div}(p_m)\}$, and $Q_\gamma = \{\bot\}$. We set $\mathsf{in} = \mathsf{in}(F)$ and $\mathsf{out} = \mathsf{out}(F)$. Let us now define the transition sets δ_0 and $\delta_{>0}$. For every subformula $G_1 \wedge G_2$ or $G_1 \vee G_2$ of F we add the following transitions to δ_0 and $\delta_{>0}$:

$$\mathsf{in}(G_1 \wedge G_2) \xrightarrow{0} \mathsf{in}(G_1),\ \mathsf{out}(G_1) \xrightarrow{0} \mathsf{in}(G_2),\ \mathsf{out}(G_2) \xrightarrow{0} \mathsf{out}(G_1 \wedge G_2)$$

$$\mathsf{in}(G_1 \vee G_2) \xrightarrow{0} \mathsf{in}(G_i),\ \mathsf{out}(G_i) \xrightarrow{0} \mathsf{out}(G_1 \vee G_2) \text{ for all } i \in \{1,2\}$$

For every variable $x_{i,r}$ we add to δ_0 and $\delta_{>0}$ the transition $\mathsf{in}(x_{i,r}) \xrightarrow{0} \mathsf{out}(x_{i,r})$. Moreover, we add to $\delta_{>0}$ the transitions $\mathsf{in}(x_{i,r}) \xrightarrow{-r} \mathrm{div}(p_i)$. The transition $\mathsf{in}(x_{i,0}) \xrightarrow{0} \mathrm{div}(p_i)$ is also added to δ_0. For the control locations $\mathrm{div}(p_i)$ we add to $\delta_{>0}$ the transitions $\mathrm{div}(p_i) \xrightarrow{-p_i} \mathrm{div}(p_i)$ and $\mathrm{div}(p_i) \xrightarrow{-1} \bot$. This concludes the description of the OCN $\mathbb{O}(F)$. Correctness of the construction can be easily checked by induction on the structure of the formula F. ∎

We are now ready to prove PSPACE-hardness of the data complexity.

Theorem 7.2. *There exists a fixed* CTL *formula of the form* $\exists\varphi_1\mathsf{U}\varphi_2$, *where* φ_1 *and* φ_2 *are* EF *formulas, for which it is* PSPACE*-complete to decide* $(T(\mathbb{O}), (q, 0)) \models \exists\varphi_1\mathsf{U}\varphi_2$ *for a given OCN* $\mathbb{O}$ *and a control location* q *of* $\mathbb{O}$.

Proof. Let us take an arbitrary language L in PSPACE. Recall from Sec. 6 that PSPACE is NC^1-serializable. Thus, there exists an NFA $A = (S, \{0,1\}, \delta, s_0, S_f)$ over the alphabet $\{0,1\}$, a polynomial $p(n)$, and a logspace-uniform NC^1-circuit family $(C_n)_{n\ge0}$, where C_n has $n + p(n)$ many inputs and one output, such that for every $x \in \{0,1\}^n$ we have:

$$x \in L \iff C_n(x, 0^{p(n)}) \cdots C_n(x, 1^{p(n)}) \in L(A), \tag{7.1}$$

where "$\cdots$" refers to the lexicographic order on $\{0,1\}^{p(n)}$. Fix an input $x \in \{0,1\}^n$. Our reduction can be split into the following five steps:

Step 1. Construct in logspace the circuit C_n. Fix the the first n inputs of C_n to the bits in x, and denote the resulting circuit by C; it has only $m = p(n)$ many inputs. Then, (7.1) can be written as

$$x \in L \iff \prod_{M=0}^{2^m-1} C(\mathrm{BIN}_m(M)) \in L(A). \tag{7.2}$$

Step 2. Compute the first m consecutive primes $p_1, \ldots, p_m$. This is possible in logspace, see e.g. [9]. Every p_i is bounded polynomially in n. Hence, every p_i can be written down in unary notation. Note that $\prod_{i=1}^{m} p_i > 2^m$ (if $m > 1$).

Step 3. Compute in logspace the circuit $B = B_m((x_{i,r})_{i\in[m],0\le r<p_i})$ from Thm. 6.1. Thus, B is a Boolean circuit of fan-in 2 and depth $O(\log(m)) = O(\log(n))$ with m output gates and $B(\mathrm{CRR}_m(M)) = \mathrm{BIN}_m(M \bmod 2^m)$ for every $0 \le M < \prod_{i=1}^{m} p_i$.

Step 4. Now we compose the circuits B and C: For every $i \in [m]$, connect the i^{th} input of the circuit $C(x_1, \ldots, x_m)$ with the i^{th} output of the circuit B. The result is a circuit with fan-in 2 and depth $O(\log(n))$. In logspace, we can unfold this circuit into a Boolean formula $F = F((x_{i,r})_{i\in[m],0\le r<p_i})$. The resulting formula (or tree) has the same depth as the circuit, i.e., depth $O(\log(n))$ and every tree node has at most 2 children. Hence, F has polynomial size. For every $0 \le M < 2^m$ we have $F(\mathrm{CRR}_m(M)) = C(\mathrm{BIN}_m(M))$ and equivalence (7.2) can be written as

$$x \in L \iff \prod_{M=0}^{2^m-1} F(\mathrm{CRR}_m(M)) \in L(A). \tag{7.3}$$

Step 5. We now apply our construction from Prop. 7.1 to the formula F. More precisely, let G be the Boolean formula $\bigwedge_{i\in[m]} x_{i,r_i}$ where $r_i = 2^m \bmod p_i$ for $i \in [m]$ (these remainders can be computed in logspace). For every 1-labeled transition $\tau \in \delta$ of the NFA A let $\mathbb{O}(\tau)$ be a copy of the OCN $\mathbb{O}(F \wedge \neg G)$. For every 0-labeled transition $\tau \in \delta$ let $\mathbb{O}(\tau)$ be a copy of the OCN $\mathbb{O}(\neg F \wedge \neg G)$. In both cases we write $\mathbb{O}(\tau)$ as $(Q(\tau), \{Q_\alpha(\tau), Q_\beta(\tau), Q_\gamma(\tau)\}, \delta_0(\tau), \delta_{>0}(\tau))$. Denote with $\mathsf{in}(\tau)$ (resp. $\mathsf{out}(\tau)$) the control location of this copy that corresponds to in (resp. out) in $\mathbb{O}(F)$. Hence, for every b-labeled transition $\tau \in \delta$ ($b \in \{0,1\}$) and every $0 \le M < \prod_{i=1}^{m} p_i$ there exists a $[\![\varphi]\!]_{T(\mathbb{O}(\tau))}$-path ($\varphi$ is from Prop. 7.1) from $(\mathsf{in}(\tau), M)$ to $(\mathsf{out}(\tau), M)$ if and only if $F(\mathrm{CRR}_m(M)) = b$ and $M \neq 2^m$.

We now define an OCN $\mathbb{O} = (Q, \{Q_\alpha, Q_\beta, Q_\gamma\}, \delta_0, \delta_{>0})$ as follows: We take the disjoint union of all the OCNs $\mathbb{O}(\tau)$ for $\tau \in \delta$. Moreover, every state $s \in S$ of the NFA A becomes a control location of $\mathbb{O}$, i.e. $Q = S \cup \bigcup_{\tau\in\delta} Q(\tau)$ and $Q_p = \bigcup_{\tau\in\delta} Q_p(\tau)$ for each $p \in \{\alpha, \beta, \gamma\}$. We add to δ_0 and $\delta_{>0}$ for every $\tau = (s, b, t) \in \delta$ the transitions $s \xrightarrow{0} \mathsf{in}(\tau)$ and $\mathsf{out}(\tau) \xrightarrow{1} t$. Then, by Prop. 7.1 and (7.3) we have $x \in L$ if and only if there exists a $[\![\varphi]\!]_{T(\mathbb{O})}$-path in $T(\mathbb{O})$ from $(s_0, 0)$ to $(s, 2^m)$ for some $s \in S_f$. Also note that there is no $[\![\varphi]\!]_{T(\mathbb{O})}$-path in $T(\mathbb{O})$ from $(s_0, 0)$ to some configuration (s, M) with $s \in S$ and $M > 2^m$. It remains to add to $\mathbb{O}$ some structure that enables $\mathbb{O}$ to check that the counter has reached the value 2^m. For this, use again Prop. 7.1 to construct the OCN $\mathbb{O}(G)$ (G is from above) and add it disjointly to $\mathbb{O}$. Moreover, add to $\delta_{>0}$ and δ_0 the transitions $s \xrightarrow{0} \mathsf{in}$ for all $s \in S_f$, where in is the in control location of $\mathbb{O}(G)$. Finally, introduce a new proposition ρ and set $Q_\rho = \{\mathsf{out}\}$, where out is the out control location of $\mathbb{O}(G)$. By putting $q = s_0$ we obtain: $x \in L$ if and only if $(T(\mathbb{O}), (q, 0)) \models \exists(\varphi \mathsf{U} \rho)$, where φ is from Prop. 7.1. This concludes the proof of the theorem. ■

By slightly modifying the proof of Thm. 7.2, one can also prove that the fixed CTL formula can chosen to be of the form $\exists\mathsf{G}\psi$, where ψ is an EF formula.

8. Two further applications: EF and one-counter Markov decision processes

In this section, we present two further applications of Thm. 6.1 to OCPs. First, we state that the combined complexity for EF over OCNs is hard for P^{NP}. For formulas represented succinctly by directed acyclic graphs this was already shown in [13]. The point here is that we use the standard tree representation for formulas.

Theorem 8.1. *It is* P^{NP}*-hard (and hence* P^{NP}*-complete by* [13]*) to check* $(T(\mathbb{O}), (q_0, 0)) \models \varphi$ *for given OCN* $\mathbb{O}$*, state* q_0 *of* $\mathbb{O}$*, and* EF *formula* φ*.*

The proof of Thm. 8.1 is very similar to the proof of Thm. 7.2, but does not use the concept of serializability. We prove hardness by a reduction from the question whether the lexicographically maximal satisfying assignment of a Boolean formula is even when interpreted as a natural number. This problem is P^{NP}-hard by [27]. At the moment we cannot prove that the data complexity of EF over OCPs is hard for P^{NP} (hardness for $\mathsf{P}^{\mathsf{NP}[\log]}$ was shown in [13]). Analyzing the proof of Thm. 8.1 in [14] shows that the main obstacle is the fact that converting from Chinese remainder representation into binary representation is not possible by uniform AC^0 circuits (polynomial size circuits of constant depth and unbounded fan-in); this is provably the case.

In the rest of the paper, we sketch a second application of our lower bound technique based on Thm. 6.1, see [14] for more details. This application concerns one-counter Markov decision processes. *Markov decision processes* (MDPs) extend classical Markov chains by allowing so called *nondeterministic vertices*. In these vertices, no probability distribution on the outgoing transitions is specified. The other vertices are called *probabilistic vertices*; in these vertices a probability distribution on the outgoing transitions is given. The idea is that in an MDP a player Eve plays against nature (represented by the probabilistic vertices). In each nondeterministic vertex v, Eve chooses a probability distribution on the outgoing transitions of v; this choice may depend on the past of the play (which is a path in the underlying graph ending in v) and is formally represented by a strategy for Eve. An MDP together with a strategy for Eve defines a Markov chain, whose state space is the unfolding of the graph underlying the MDP. Here, we consider infinite MDPs, which are finitely represented by OCPs; this formalism was introduced in [6] under the name *one-counter Markov decision process* (OC-MDP). With a given OC-MDP $\mathcal{A}$ and a set R of control locations of the OCP underlying $\mathcal{A}$ (a so called *reachability constraint*), two sets were associated in [6]: $\text{ValOne}(R)$ is the set of all vertices s of the MDP defined by $\mathcal{A}$ such that for every $\epsilon > 0$ there exists a strategy σ for Eve under which the probability of finally reaching from s a control location in R and at the same time having counter value 0 is at least $1 - \varepsilon$. $\text{OptValOne}(R)$ is the set of all vertices s of the MDP defined by $\mathcal{A}$ for which there exists a specific strategy for Eve under which this probability is 1. It was shown in [6] that for a given OC-MDP $\mathcal{A}$, a set of control locations R, and a vertex s of the MDP defined by $\mathcal{A}$, the question if $s \in \text{OptValOne}(R)$ is PSPACE-hard and in $\mathsf{EXPTIME}$. The same question for $\text{ValOne}(R)$ instead of $\text{OptValOne}(R)$ was shown to be hard for each level of the Boolean hierarchy BH, which is a hierarchy of complexity classes between NP and $\mathsf{P}^{\mathsf{NP}[\log]}$. By applying our lower bound techniques (from Thm. 7.2) we can prove the following.

Theorem 8.2. *Membership in ValOne*(R) *is* PSPACE*-hard.*

As a byproduct of our proof, we also reprove PSPACE-hardness for $\text{OptValOne}(R)$. It is open, whether $\text{ValOne}(R)$ is decidable; the corresponding problem for MDPs defined by pushdown processes is undecidable [11].

References

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[2] C. Baier and J. P. Katoen. *Principles of Model Checking*. MIT Press, 2009.

[3] P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM J. Comput.*, 15(4):994–1003, 1986.

[4] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. CONCUR'97*, LNCS 1243, 135–150. Springer, 1997.

[5] L. Bozzelli. Complexity results on branching-time pushdown model checking. *T. Comput. Sci.*, 379:286–297, 2007.

[6] T. Brazdil, V. Brozek, K. Etessami, A. Kucera, and D. Wojtczak. One-counter markov decision processes. *Proc. SODA 2010*, 863–874. SIAM, 2010.

[7] T. Cachat. Uniform solution of parity games on prefix-recognizable graphs. *ENTCS*, 68(6):71–84, 2002.

[8] J.-Y. Cai and M. Furst. PSPACE survives constant-width bottlenecks. *Internat. J. Found. Comput. Sci.*, 2(1):67–76, 1991.

[9] A. Chiu, G. Davida, and B. Litow. Division in logspace-uniform NC^1. *RAIRO Inform. Théor. Appl.*, 35(3):259–275, 2001.

[10] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. CAV 2000*, LNCS 1855, 232–247. Springer, 2000.

[11] K. Etessami and M. Yannakakis. Recursive markov decision processes and recursive stochastic games. In *Proc. ICALP 2005*, LNCS 3580, 891–903. Springer, 2005.

[12] S. Göller and M. Lohrey. Infinite state model-checking of propositional dynamic logics. In *Proc. CSL 2006*, LNCS 4207, 349–364. Springer, 2006.

[13] S. Göller, R. Mayr, and A. W. To. On the computational complexity of verifying one-counter processes. In *Proc. LICS 2009*, 235–244. IEEE Computer Society Press, 2009.

[14] S. Göller, M. Lohrey. Branching-time model checking of one-counter processes. `http://arxiv.org/abs/0909.1102`.

[15] C. Haase, S. Kreutzer, J. Ouaknine, and J. Worrell. Reachability in succinct and parametric one-counter automata. In *Proc. CONCUR'09*, LNCS 5710, 369–383. Springer, 2009.

[16] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proc. 8th Annual Structure in Complexity Theory Conference*, 200–207. IEEE Computer Society Press, 1993.

[17] W. Hesse, E. Allender, and D. A. M. Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. System Sci.*, 65:695–716, 2002.

[18] M. Holzer. On emptiness and counting for alternating finite automata. In *Proc. DLT 1995*, 88–97. Wo. Scient., 1996.

[19] P. Jančar and Z. Sawa. A note on emptiness for alternating finite automata with a one-letter alphabet. *I. P. L.*, 104(5):164–167, 2007.

[20] M. Nair. On Chebyshev-type inequalities for primes. *Amer. Math. Monthly*, 89(2):126–129, 1982.

[21] N. Piterman and M. Y. Vardi. Global model-checking of infinite-state systems. In *Proc. CAV 2004*, LNCS 3114, 387–400. Springer, 2004.

[22] O. Serre. Note on winning positions on pushdown games with ω-regular conditions. *I. P. L.*, 85(6):285–291, 2003.

[23] O. Serre. Parity games played on transition graphs of one-counter processes. In *Proc. FOSSACS 2006*, LNCS 3921, 337–351. Springer, 2006.

[24] D. Thérien and T. Wilke. Temporal logic and semidirect products: An effective characterization of the until hierarchy. In *Proc. FOCS '96*, 256–263. IEEE Computer Society Press, 1996.

[25] A. W. To. Model checking FO(R) over one-counter processes and beyond. In *Proc. CSL 2009*, LNCS 5771, 485–499. Springer, 2009.

[26] H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.

[27] K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoret. Comput. Sci.*, 51:53–80, 1987.

[28] I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Proc. FSTTCS 2000*, LNCS 1974, 127–138. Springer, 2000.

[29] I. Walukiewicz. Pushdown processes: games and model-checking. *Inform. and Comput.*, 164(2):234–263, 2001.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 417-428
www.stacs-conf.org

EVOLVING MULTIALGEBRAS UNIFY ALL USUAL SEQUENTIAL COMPUTATION MODELS

SERGE GRIGORIEFF[1] AND PIERRE VALARCHER[2]

[1] LIAFA, CNRS & Université Paris Diderot - Paris 7, Case 7014, 75205 Paris Cedex 13
E-mail address: seg@liafa.jussieu.fr
URL: http://www.liafa.jussieu.fr

[2] LACL, Université de Paris Est, IUT Fontainebleau/Sénart, Route de Hourtault 77300 Fontainebleau
E-mail address: valarcher@univ-paris12.fr
URL: http://lacl.univ-paris12.fr/valarcher/

Abstract. It is well-known that Abstract State Machines (ASMs) can simulate "step-by-step" any type of machines (Turing machines, RAMs, etc.). We aim to overcome two facts: 1) simulation is not identification, 2) the ASMs simulating machines of some type do not constitute a natural class among all ASMs. We modify Gurevich's notion of ASM to that of EMA ("Evolving MultiAlgebra") by replacing the program (which is a syntactic object) by a semantic object: a functional which has to be very simply definable over the static part of the ASM. We prove that very natural classes of EMAs correspond via "literal identifications" to slight extensions of the usual machine models and also to grammar models. Though we modify these models, we keep their computation approach: only some contingencies are modified.
Thus, EMAs appear as the mathematical model unifying all kinds of sequential computation paradigms.

Contents

Key words and phrases: Abstract state machines; Models of machines; Computability; Universality; Logic in computer science; Theory of algorithms.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2473

1. Introduction

What we prove in this paper. The fact that Abstract State Machines (ASMs) can strict lock-step (i.e. "step-by-step ") simulate any type of machines (Turing machines, stack automata, RAM, etc) and grammars was shown long ago by Gurevich [10, 6]. A systematic study is also done in Börger [2]. A tighter notion of simulation is also valid as shown in Blass, Dershowitz & Gurevich [1].
The questions we consider in this paper are the following:

(Q1) *Can we replace strict lock-step simulation by literal identity (up to a simple change of view)?*

(Q2) *Given a computation model $\mathcal{C}$, is it possible to get a natural characterization of the class of ASMs which are equivalent to machines in $\mathcal{C}$?*

As far as we know, up to now, there is only one isolated answer which is about question (Q2): Gurevich & al. [6] proved that Schönhage Storage Modification Machines correspond exactly (for strict lock-step equivalence) to ASMs with unary functions only.
We bring positive answers to both questions for the diverse usual computation models $\mathcal{C}$ (Turing machines, stack automata, RAMs, Schönhage Machines, Chomsky type 0 grammars, etc.) slightly extended to models $\mathcal{C}^+$ using a tailored version of ASMs which (resurrecting Gurevich's original name for ASMs) we call *Evolving Multialgebras* (EMAs). These answers have the following remarkably simple form:

Theorem 1.1. *There exists a family of EMA static parts $\mathcal{M}$ (fixed semantical feature) and a family of dynamic signatures $\mathcal{S}$ (fixed syntactical feature) such that, letting $\mathcal{E}_{M,S}$ be the family of EMAs with static part in $\mathcal{M}$ and dynamic signature in $\mathcal{S}$,*
- any computation device in $\mathcal{C}^+$ is literally identical to some EMA in $\mathcal{E}_{\mathcal{M},\mathcal{S}}$,
- this "literal identity" correspondence is a bijection from $\mathcal{C}^+$ onto $\mathcal{E}_{\mathcal{M},\mathcal{S}}$.

Of course, *literal identity* is not a formal notion. What we mean is as follows: the diverse components of a computation device in $\mathcal{C}^+$ are in one-one correspondance with the diverse components of the associated EMA, and this correspondance is an identity up to a change of perspective (for instance, a "physical" bi-infinite tape will be considered to be identical to the mathematical set $\mathbb{Z}$ of integers).

Remark 1.2. 1. This theorem is indeed a schema: one theorem per computation model. We have proved it for a variety of usual sequential computation models (cf. [5]).
2. As said above, the diverse instances of Theorem 1.1 are proved for slight extension $\mathcal{C}^+$ of the usual computation models $\mathcal{C}$. In all cases, $\mathcal{C}^+$ can be viewed as $\mathcal{C}$ considered with different time units: for any $k \geq 1$, a device $\mathcal{M}$ in $\mathcal{C}$ is seen as a device $\mathcal{M}^{(k)}$ in $\mathcal{C}^+$ in which one step of $\mathcal{M}^{(k)}$ corresponds to k successive steps of $\mathcal{M}$ (or $< k$ successive steps in case the last of these steps has no successor).
3. Considering another presentation of $\mathcal{C}^+$, one can also view it as $\mathcal{C}$ in which some contingencies have been removed (for instance, the read/write head will be able to scan a window of cells instead of a single cell) but the computational paradigm has been preserved: local computation and a particular topology of data storage for Turing machines, indirect addressing of registers for random access machines, etc. In our opinion, the classes $\mathcal{C}^+$ are the *right ones* to carry the diverse computation paradigms.
4. In fact, contingencies can also be captured by families of EMAs with more technical definitions (cf. [5]): we loose the remarkable simplicity of the above families $\mathcal{E}_{\mathcal{M},\mathcal{S}}$.
5. This theorem schema strengthens Gurevich's claim that ASMs constitute the natural

mathematical modelization of algorithms: *EMAs (which are a variant of ASMs) appear as the computation model which unifies all usual sequential computation paradigms.*

About the proof. No surprise, the proof of Theorem 1.1 for a particular $\mathcal{C}$ involves the particular features of the class $\mathcal{C}$. Thus, the claim (point 5 in Remark 1.2) that Theorem 1.1 is true for extensions $\mathcal{C}^+$ of every usual sequential computation model $\mathcal{C}$ cannot be proved but only be supported by proved instances for a variety of classes $\mathcal{C}$.

As for the common features to all such proofs, they come from an analysis of what precludes positive solutions to questions (Q1) and (Q2). Let us list some of the difficulties which are met. Some are easy to solve, other ones force to adequately tailor the definition of ASMs (as that of EMAs) and those of the usual computation models.

(1) An ASM program mimicking the transition function δ of a Turing machine is a description of δ. Since there are many distinct descriptions of the same δ, there are many ASMs which tightly simulate the same Turing machine. Thus, surprisingly as it is, looking at this component – transition functions –, ASMs are less abstract than Turing machines. Somehow, there is an extra operational feature in ASMs: the operational way to use δ is not part of the formalization of Turing machines.
This is why we modify ASMs to EMAs: *Evolving Multialgebras.* The notion of EMA is that of ASM in which the program (a syntactic object) is replaced by a semantic object: a (very simply definable) functional operating on the function sets over the ASM domain. It is then more natural to break the universe of an ASM into its natural parts: this allows a very useful rudimentary typing of elements and functions.

(3) Again considering Turing machines, an ASM simulates the tape by the set $\mathbb{Z}$ of all integers and the moves of the head by the successor and predecessor operations on $\mathbb{Z}$. Terms in the ASM logical language allow to name the i-th successor and the i-th predecessor. Thus, we cannot avoid the ASM program to move the head more than one cell left or right unless we constrain terms in ASM programs to be of a simple form (somewhat "flat"). Which would put technicalities to any positive answer to question (Q2). This is why we consider slight extensions of the machine models which allow the read/write head to scan a window of cells rather than only one cell and to move in a window. This is a kind of extra capability which is much like allowing several tapes or several heads. Though it does modify the model, it does preserves its core feature: successive local actions.

(4) For machine models having programs like RAMs and SMM (Schönhage Storage Modication Machines), there are two slight modifications. First, allow bounded blocks of parallel and/or successive actions. Second, remove the program and the program counter in favor of a transition function (much in the vein of Turing machines) which, though operating on an infinite set (the contents of the accumulator and of the addressed registers in the case of RAMs) is very simply definable in terms of the original program. Thus, we replace an operational item (the program) by a denotational one (the transition function). Again, though it does modify the model, it does preserves its core feature: indirect addressing (for RAMs), dynamic storage topology (for Schönhage pointer machines).

EMAs versus ASMs. In our opinion, ASMs and EMAs are complementary models. EMAs generalize any type of machine: it is the unification model. As for ASMs, they are closer to programming. Indeed, the functioning of a EMA goes through the iteration of a functional. To program an EMA, we need to add some operational information on how to use this functional and this leads back to a program, hence to an ASM. . . Thus, ASMs are

EMAs plus the instructions for using the functional: ASMs refine EMAs (in the sense of software engineering) and EMAs are a (more) abstract version of ASMs.

2. From ASMs to EMA s: the deterministic case

2.1. How EMAs differ from ASMs

We detail the diverse features which are peculiar to EMAs.

A functional in place of a program. As said in the introduction, the main difference between evolving multialgebras and Gurevich's ASMs is as follows: *the program (i.e. a syntactic object) of an ASM is replaced by a functional (i.e. a semantic object)* which does exactly what the program tells to do. Thus, an operational feature is removed.

Multi-domains and multialgebras. The above modification leads to another very minor one, really kind of "semantic sugar": *the universe of an ASM is broken into its natural constituents and becomes a multi-domain.* The reason for such multialgebras is that they make it possible to type the symbols of the signature as functions (or elements) between the diverse sets of the multi-domain.

Multialgebra operations with values in products of domains. Set theoretically, a map $F : A \to B \times C$ is identified with the pair of its component maps (F_B, F_C) where $F_B : A \to B$ and $F_C : A \to C$. We do view such an F as the pair (F_B, F_C) plus a correlation condition: *one cannot fire one of these two component maps without firing the other one, and both have to be fired on the same argument.*
We allow operations in the multialgebra to take values in products of domains. The above condition leads to a notion of multiterms and a constraint in the definition of formulas associated to the signature of an EMA. It is used in §**??** to deal with Schönhage machines.

Halt/Fail and EMA status. In EMAs, the ASM program is replaced by the functional which does exactly what the program tells to do. There are still the questions:
- is the functional to be applied or not on given arguments?
- if not, does it "halts and accepts" or "halts and rejects" or "get stuck"?
To deal with the three first alternatives, EMAs have a three valued dynamic component: the status. Of course, there is no formal component carrying the information "stuck".

Inputs and ASMs. In most presentations, Gurevich does not give any formal status to inputs (his paper [4] with Dershowitz being an exception). When dealing with question (Q2) it turns out that it is important to give a formal status to inputs. This is the case for EMA characterizations of machines having some read-only tapes (e.g., finite automata).
We consider that *inputs appear in two ways:*
- as values of some particular static symbols,
- as initial values of dynamic symbols.

2.2. Deterministic Evolving MultiAlgebras

Definition 2.1. Let $n \geq 1$ and $\mathcal{D} = (D_i)_{i=1,\ldots,n}$ be a sequence of n non empty sets (which we call an n-multiset). An n-sort type is a triple (k, α, ℓ) where $k \in \mathbb{N}$, $\ell \in \{1, \ldots, n\}$ and α is a map $\{1, \ldots, k\} \to \{1, \ldots, n\}$. Its associated $\mathcal{D}$-type $(k, \alpha, \ell)_{\mathcal{D}}$ is the family of all partial functions $D_{\alpha(1)} \times \ldots \times D_{\alpha(k)} \to D_\ell$. A $\mathcal{D}$-type is functional if $k \geq 1$. In case $k = 0$, the $\mathcal{D}$-type $(0, \emptyset, \ell)_{\mathcal{D}}$ is the family of partial functions $\{\emptyset\} \to D_\ell$, i.e. the set of "partial elements" of D_ℓ, i.e. D_ℓ augmented with an "undefined element".
Intuition: there are k arguments, α gives their types, and ℓ is the type of the range.
Typed ground terms and their types are defined in the obvious way.

Multialgebras. The notion of multisort algebra is a direct extension to multiset domains of the usual notion of algebra of partial functions on a unique domain.

Definition 2.2 (Multialgebras)**.** Let $n \geq 1$ and $\mathcal{S}$ be an n-sort typed signature containing function symbols $\varphi_1, \ldots, \varphi_p$. An $\mathcal{S}$-multialgebra $\mathcal{A}$ is an n-multiset $\mathcal{D} = (D_i)_{i=1,\ldots,n}$ endowed with partial functions $F_1, \ldots, F_p$ which interpret the symbols φ_i's (Care: arity 0 symbols with type D_i are interpreted by elements of D_i but can also be undefined).
If defined, the value, relative to $\mathcal{A}$, of a ground $\mathcal{S}$-term t is denoted by $[\![t]\!]_{\mathcal{A}}$ (it is an element of some D_i).

Semialgebraic functionals. Semialgebraic functionals are those which can be described by ASM programs. They modify the interpretations in the multialgebra of constant and functions symbols. For function symbols, this modification affects the values of only finitely many points in the domain. These points and the associated new values of the argument are given by ground $\mathcal{S}$-terms. As in ASMs programs, there is a disjunction of cases for the choice of the affected points and their associated new values.

First, a convenient notion.

Definition 2.3 (The $\oplus$ operation)**.** Let F, G be partial functions $X_1 \times \ldots \times X_k \to Y$ and $Z \subseteq X_1 \times \ldots \times X_k$. We define the partial function $F \oplus_Z G$ as follows:

$$\begin{array}{rcl} \mathtt{Domain}(F \oplus_Z G) & = & (\mathtt{Domain}(F) \setminus Z) \cup (\mathtt{Domain}(G) \cap Z) \\ (F \oplus_Z G)(\vec{x}) & = & \left\{ \begin{array}{lll} F(\vec{x}) & \mathit{if} & \vec{x} \notin Z \\ G(\vec{x}) & \mathit{if} & \vec{x} \in Z \end{array} \right. \end{array}$$

In case $p = 0$, F, G are "partial elements" of Y and $Z \subseteq \{\emptyset\}$ and $F \oplus_Z G = F$ if $Z = \emptyset$ and $F \oplus_Z G = G$ if $Z = \{\emptyset\}$.

Definition 2.4 (**Semialgebraic functionals**)**.** Let
- $\mathcal{D} = (D_i)_{i=1,\ldots,n}$ be an n-multiset,
- $\mathcal{S}$ be an n-sort typed signature containing function symbols $\varphi_1, \ldots, \varphi_p$,
- $\mathcal{A}$ be a multialgebra with signature $\mathcal{S} \setminus \{\varphi_1, \ldots, \varphi_p\}$ on $\mathcal{D}$,
- $\mathcal{F}_1, \ldots, \mathcal{F}_p$ be the $\mathcal{D}$-types associated to $\varphi_1, \ldots, \varphi_p$,
- $m \in \{1, \ldots, p\}$ and (k, α, ℓ) be the n-sort type of φ_m.
- $\mathcal{T}_i$ be the family of ground $\mathcal{S}$-terms of type D_i,

For any p-tuple of functions $\vec{F} = (F_1, \ldots, F_p) \in \mathcal{F}_1 \times \ldots \times \mathcal{F}_p$, let us denote by $\mathcal{A}(\vec{F})$ the multialgebra $\mathcal{A}$ expanded to the signature $\mathcal{S}$ in which the φ_i's are interpreted by the F_i's.
A partial functional $\Phi : \prod_{j=1,\ldots,p} \mathcal{F}_j \longrightarrow \mathcal{F}_m$ is $(\mathcal{S}, \mathcal{A})$-semialgebraic if there exists a map $\beta : \mathtt{Bool}^q \to \mathfrak{P}_{\mathit{fin}}(\mathcal{T}_{\alpha(1)} \times \ldots \times \mathcal{T}_{\alpha(k)} \times \mathcal{T}_\ell)$ (where $\mathfrak{P}_{\mathit{fin}}(X)$ is the family of finite subsets of

X) and ground $\mathcal{S}$-terms $t_1,\dots,t_q$, $t'_1\dots,t'_q$ such that, for any $\vec{F}\in\mathcal{G}_1\times\dots\times\mathcal{G}_p$,

$$\Phi(\vec{F}) \text{ is defined if and only if}$$
$$\begin{cases} (a) & \text{all } [\![t_i]\!]_{\mathcal{A}(\vec{F})}\text{'s, } [\![t'_i]\!]_{\mathcal{A}(\vec{F})}\text{'s are defined} \\ (b) & \forall (u_1,\dots,u_k,v)\in\beta(\dots,[\![t_i]\!]_{\mathcal{A}(\vec{F})}=[\![t'_i]\!]_{\mathcal{A}(\vec{F})},\dots)\ \text{all } [\![u_j]\!]_{\mathcal{A}(\vec{F})}\text{'s are defined} \\ (c) & \forall (\vec{u},v),(\vec{w},z)\in\beta(\dots,[\![t_i]\!]_{\mathcal{A}(\vec{F})}=[\![t'_i]\!]_{\mathcal{A}(\vec{F})},\dots)\quad [\![u_j]\!]_{\mathcal{A}(\vec{F})}\neq[\![w_j]\!]_{\mathcal{A}(\vec{F})} \text{ for some } j \end{cases}$$

$$\Phi(\vec{F}) = F_m\oplus_Z G \text{ where}$$
$$Z=\{([\![u_1]\!]_{\mathcal{A}(\vec{F})},\dots,[\![u_k]\!]_{\mathcal{A}(\vec{F})})\mid \exists v\ (\vec{u},v)\in\beta(\dots,[\![t_i]\!]_{\mathcal{A}(\vec{F})}=[\![t'_i]\!]_{\mathcal{A}(\vec{F})},\dots)\}$$
$$G=\{([\![u_1]\!]_{\mathcal{A}(\vec{F})},\dots,[\![u_k]\!]_{\mathcal{A}(\vec{F})},[\![v]\!]_{\mathcal{A}(\vec{F})})\mid (\vec{u},v)\in\beta(\dots,[\![t_i]\!]_{\mathcal{A}(\vec{F})}=[\![t'_i]\!]_{\mathcal{A}(\vec{F})},\dots)\}$$

The tuple $(\beta,t_1,\dots,t_q,t'_1\dots,t'_q)$ is called a presentation of Φ.

For $I\subseteq\{1,\dots,p\}$, a functional $\Psi:\prod_{j=1,\dots,p}\mathcal{F}_j\longrightarrow\prod_{m\in I}\mathcal{F}_m$ is $(\mathcal{S},\mathcal{A})$-semialgebraic if so are all its components.

Remark 2.5. Condition (a) in Definition 2.4 insures that all equality tests $t_i=t'_i$ can be achieved. Conditions (b) and (c) insure that, in equality $\Phi(\vec{F})=F_m\oplus_Z G$, the finite set Z can be computed and G is a functional graph.
We do not require the $[\![v]\!]_{\mathcal{A}(\vec{F})}$'s to be defined (i.e. $\mathtt{Domain}(G)=Z$): though this is incompatible with a call by value strategy, it makes sense with a call by name strategy.

Definition 2.6 (Deterministic EMAs). A deterministic evolving multialgebra (EMA) is a tuple $\mathcal{A}=(n;\ \mathcal{S}_{\rm sta},\mathcal{S}^{\rm sta}_{\rm input},\mathcal{S}^{\rm dyn}_{\rm input},\mathcal{S}_{\rm dyn};\ \mathcal{D};\ \mathcal{M}_{\rm sta},\mathcal{M}_{\rm ini};\ \Phi)$ consisting of the following items.

- An n-multiset $\mathcal{D}=(D_i)_{i=1,\dots,n}$ such that $D_n=\{\text{go},\text{acc},\text{rej}\}$.
Intuition. Sets $D_1,\dots,D_{n-1}$ are the $n-1$ different sorts of objects and $D_n=\{go,acc,rej\}$ is the set of possible statuses of the (evolving) multialgebra during the run: "go on", "halt and accept", "halt and reject".
- Four disjoint n-sort typed finite signatures $\mathcal{S}_{\rm sta},\mathcal{S}^{\rm sta}_{\rm input},\mathcal{S}^{\rm dyn}_{\rm input},\mathcal{S}_{\rm dyn}$ and two structures $\mathcal{M}_{\rm sta},\mathcal{M}_{\rm ini}$ with respective signatures $\mathcal{S}_{\rm sta},\mathcal{S}_{\rm dyn}$. There is only one symbol $\mathfrak{s}$ which involves the sort n : it is a constant of type D_n in $\mathcal{S}^{\rm dyn}_{\rm input}$.
Intuition. $\mathcal{M}_{sta}$ is the static framework on $\mathcal{D}$ which remains fixed during any run. $\mathcal{S}^{sta}_{input}$ is the signature for the static part of the input: its interpretation remains fixed (hence accessible) during a run. $\mathcal{S}^{dyn}_{input}$ is the signature for the dynamic part of the input: its interpretation can be modified (hence become inaccessible) during a run. $\mathcal{M}_{ini}$ initializes the part of the dynamic environment which is not initialized by the input. The interpretation of $\mathfrak{s}$ represents the status of the multialgebra.
- Let $\mathcal{S}=\mathcal{S}_{\rm sta}\cup\mathcal{S}^{\rm sta}_{\rm input}\cup\mathcal{S}^{\rm dyn}_{\rm input}\cup\mathcal{S}_{\rm dyn}$. Φ is a $(\mathcal{S},\mathcal{M}_{\rm sta})$-semialgebraic partial functional
$$\Phi:\left(\prod_{\varphi\in\mathcal{S}^{\rm sta}_{\rm input}}\mathcal{F}_\varphi\right)\times\left(\{\text{go}\}\times\prod_{\varphi\in(\mathcal{S}_{\rm dyn}\cup\mathcal{S}^{\rm dyn}_{\rm input})\setminus\{\mathfrak{s}\}}\mathcal{F}_\varphi\right)\longrightarrow\prod_{\varphi\in\mathcal{S}_{\rm dyn}\cup\mathcal{S}^{\rm dyn}_{\rm input}}\mathcal{F}_\varphi$$
where $\mathcal{F}_\varphi$ denotes the semantic type of the function symbol φ. In particular, Φ rules the evolution of the status. The sole status which can be an argument of Φ is "go": a multialgebra with status "acc" or "rej" is halted and does not evolve any more. However, in the image of Φ the status can take any value.

A state of $\mathcal{A}$ is any multialgebra on $\mathcal{D}$ with signature $\mathcal{S}$ which expands $\mathcal{M}_{\text{sta}}$.

Definition 2.7 (Runs of deterministic EMAs). We keep the notations of Definition 2.6. A run of $\mathcal{A}$ is a sequence $(\mathcal{M}_t)_{t\in I}$ of states of $\mathcal{A}$ such that
- I is a finite or infinite non empty initial segment of $\mathbb{N}$,
- $[\![\theta]\!]_{\mathcal{M}_0} = [\![\theta]\!]_{\mathcal{M}_{\text{ini}}}$ for all $\theta \in \mathcal{S}_{\text{dyn}}$,
- If $t \in I$ then $[\![\theta]\!]_{\mathcal{M}_t} = [\![\theta]\!]_{\mathcal{M}_0}$ for all $\theta \in \mathcal{S}^{\text{sta}}_{\text{input}}$,
- If $t \in I$ then $t+1$ is in I if and only if $[\![\mathfrak{s}]\!]_{\mathcal{M}_t} = \text{go}$ and Φ is defined on $([\![\varphi]\!]_{\mathcal{M}}\})_{\varphi\in\mathcal{S}\setminus\mathcal{S}_{sta}}$,
- If $t+1 \in I$ then $([\![\theta]\!]_{\mathcal{M}_{t+1}})_{\theta\in\mathcal{S}_{\text{dyn}}\cup\mathcal{S}^{\text{dyn}}_{\text{input}}} = \Phi(([\![\varphi]\!]_{\mathcal{M}_t}\})_{\varphi\in\mathcal{S}\setminus\mathcal{S}_{sta}})$.

In particular, if $[\![\mathfrak{s}]\!]_{\mathcal{M}_0} \neq \text{go}$ then $I = \{0\}$. Also, if $t+1 \in I$ then $[\![\mathfrak{s}]\!]_{\mathcal{M}_t} = \text{go}$.

3. Turing machines

In order to *identify* Turing machines with a simple class of EMAs, we introduce a slight variant of Turing machines, which we call "window Turing machines": 1) the head is allowed to scan a small window instead of a single cell, and to move inside a window in a single step, 2) halting (be it accepting or rejecting) is not related to the current state but to the current local configuration: the state plus the contents of the scanned window.

Definition 3.1. A deterministic k-window n-tape (bi-infinite tapes) Turing machine is a tuple $(n, k, \Sigma = \{\sigma_0, \ldots, \sigma_{s-1}\}, Q = \{q_0, \ldots, q_{r-1}\}, F^+, F^-, \delta, \omega_i, \mu_i)_{i=1,\ldots,n}$ where, for $i = 1, \ldots, n$,
- Σ and Q are finite sets (the alphabet and the set of states),
- $F^+, F^- \subseteq Q \times \Sigma^{n(2k+1)}$ (accepting/rejecting final local configurations),
- $\delta : Q \times \Sigma^{n(2k+1)} \to Q$ (state transition),
- $\tau_i : Q \times \Sigma^{n(2k+1)} \to \Sigma^{n(2k+1)}$ (read/write on tape i),
- $\mu_i : Q \times \Sigma^{n(2k+1)} \to \{-k, \ldots, -1, 0, 1, \ldots, k\}$ (move on tape i).

On each tape, the head scans the cell on which it is positioned and the k cells to the left and the k cells to the right, a total of $2k+1$ cells. The argument of type $\Sigma^{n(2k+1)}$ in δ, ω_i, μ_i is the contents of the $n(2k+1)$ cells scanned on the n tapes. The effect of a transition is to change the state according to δ, to modify the contents of the scanned cells of tape i according to ω_i and to move its head according to μ_i.
The notions of run, halt, acceptance and rejection are defined as usual.

Remark 3.2. Usual deterministic n-tape Turing machines are the 1-window ones.

Definition 3.3 (The class of EMAs for Turing machines). We denote by $\mathcal{C}^{(n)}_{wT}$ the class of EMAs $\mathcal{A} = (n+3;\ \mathcal{S}_{\text{sta}}, \mathcal{S}^{\text{sta}}_{\text{input}}, \mathcal{S}^{\text{dyn}}_{\text{input}}, \mathcal{S}_{\text{dyn}};\ \mathcal{D};\ \mathcal{M}_{\text{sta}}, \mathcal{M}_{\text{ini}};\ \Phi)$ which satisfy the following conditions for some $r, s \in \mathbb{N}$ (for clarity, we abusively denote by the same letter static constant symbols and the elements which interpret them in the structure $\mathcal{D}$).

(1) The multidomain of $\mathcal{A}$ is $\mathcal{D} = (\mathbb{Z}^{(1)}, \ldots, \mathbb{Z}^{(n)}, Q, \Sigma, \mathfrak{S})$ where the $\mathbb{Z}^{(i)}$'s are fixed pairwise disjoint copies of $\mathbb{Z}$ (for instance, $\mathbb{Z}^{(i)} = \mathbb{Z} \times \{i\}$), Q, Σ are finite sets with r, s elements respectively, and $\mathfrak{S} = \{\text{go}, \text{acc}, \text{rej}\}$.

(2) The static framework signature $\mathcal{S}_{\text{sta}}$ contains r constants $q_0, \ldots, q_{r-1}$ of type Q, s constants $\sigma_0, \ldots, \sigma_{s-1}$ of type Σ and three constants go, acc, rej of type $\mathfrak{S}$ which are interpreted in the obvious way in $\mathcal{M}_{\text{sta}}$. It also contains, for each $i = 1, \ldots, n$, two unary functions

symbols $\mathit{Succ}^{(i)}, \mathit{Pred}^{(i)}$ of type $\mathbb{Z}^{(i)} \to \mathbb{Z}^{(i)}$ which are interpreted in $\mathcal{M}_{\text{sta}}$ as the successor and predecessor functions in $\mathbb{Z}^{(i)}$.

(3) The signature $\mathcal{S}^{\text{sta}}_{\text{input}}$ is empty.

(4) The signature $\mathcal{S}_{\text{dyn}}$ (for the dynamic environment non initialized by the input) contains, for each $i = 1, \ldots, n$, one constant $\text{pos}^{(i)}$ of type $\mathbb{Z}^{(i)}$ one constant q of type Q, and one constant $\mathfrak{s}$ of type $\mathfrak{S}$, which are respectively interpreted in $\mathcal{M}_{\text{ini}}$ as 0, q_0 and go.

(5) The signature $\mathcal{S}^{\text{dyn}}_{\text{input}}$ (for the dynamic environment initialized by the input) contains, for each $i = 1, \ldots, n$, one unary function $c^{(i)}$ of type $\mathbb{Z}^{(i)} \to \Sigma$.

Thus, the EMAs in $\mathcal{C}^{(n)}_{wT}$ are defined as those having particular signature, multidomain, static framework and initialization of some dynamic symbols with no condition on the functional Φ (other than its semialgebraicity).

Theorem 3.4 (**EMA representation theorem for Turing machines**).
Any deterministic n-tape window Turing machine is literally identical *to some EMA in the class $\mathcal{C}^{(n)}_{wT}$. Conversely, any EMA in $\mathcal{C}^{(n)}_{wT}$ is* literally identical *to some deterministic n-tape window Turing machine.*

Proof. The argument is based on the following literal identifications between the components of a Turing machine (TM) and the interpretations of symbols of the EMA signature:

(1) (TM) i-th tape and the way the read/write head moves on it.
(EMA) the copy $\mathbb{Z}^{(i)}$ of $\mathbb{Z}$ structured as $\langle \mathbb{Z}^{(i)}, \mathit{Succ}^{(i)}, \mathit{Pred}^{(i)} \rangle$.

(2) (TM) diverse states and letters.
(EMA) interpretations of the static symbols $q_0, \ldots, q_{r-1}$ and $\sigma_0, \ldots, \sigma_{s-1}$.

(3) (TM) current state, positions of the n heads and contents of the n tapes.
(EMA) current interpretations of the dynamic symbols q, $\mathit{pos}^{(i)}$, $c^{(i)}$.

(4) (TM) non final or final accepting/rejecting character of the current state.
(EMA) current interpretation of the dynamic symbol $\mathfrak{s}$.

(5) (TM) transition function.
(EMA) semialgebraic functional.

(6) (TM) initial configuration.
(EMA) interpretations of the c_i's in the initial multialgebra and of $\mathcal{S}^{\text{dyn}}_{\text{dyn}}$ in $\mathcal{M}_{\text{ini}}$.

The non trivial identifications are those of points 4 and 5.
Keeping the notations of Definition 2.4, let $(\beta_\varphi, t_{1,\varphi}, \ldots, t_{q_\varphi,\varphi}, t'_{1,\varphi} \ldots, t'_{q_\varphi,\varphi})_{\varphi \in \mathcal{S}^{\text{int}}_{\text{dyn}}}$ be a presentation of the semialgebraic functional Φ of an EMA:

$$\beta_\varphi : \mathtt{Bool}^{q_\varphi} \to \mathfrak{P}_{\mathit{fin}}(\mathcal{T}_{\alpha_\varphi(1)} \times \ldots \times \mathcal{T}_{\alpha_\varphi(k_\varphi)} \times \mathcal{T}_{\ell_\varphi})$$

Observe that terms of type $\mathbb{Z}^{(j)}$ are of the form $\xi_1(\xi_2(\ldots))(\text{pos}^{(j)})$ where the ξ_k's are $\mathit{Succ}^{(j)}$ or $\mathit{Pred}^{(j)}$. Let k be the maximum value of the $|\xi_1(\xi_2(\ldots))(0)|$ for all terms of type some $\mathbb{Z}^{(j)}$ which is among the $t_{i,\varphi}, t'_{i,\varphi}$ or among the finite sets given by the β_φ's.

First, let us look at the equalities $t_{i,\varphi} = t'_{i,\varphi}$ which govern the domain of Φ.

• If $t_{i,\varphi}, t'_{i,\varphi}$ have type $\mathbb{Z}^{(j)}$ then, as said above, they are of the form $\xi_1(\xi_2(\ldots))(\text{pos}^{(j)})$. Hence any equality $t_{i,\varphi} = t'_{i,\varphi}$ is trivially true or false independently of the current value of $\text{pos}^{(j)}$.

If $t_{i,\varphi}, t'_{i,\varphi}$ have type $\mathfrak{S}$ then they are of the form $\mathfrak{s}$ or go, acc, rej. Since Φ and β are restricted

to values where $\mathfrak{s} = \text{go}$, all possible equalities are trivial.
Thus, we can suppose that there is no term with type $\mathbb{Z}^{(j)}$ or $\mathfrak{S}$ among the $t_{i,\varphi}, t'_{i,\varphi}$'s.

- If $t_{i,\varphi}, t'_{i,\varphi}$ have type Q then they are of the form q or q_j $(j = 0, \ldots, r-1)$. Since any equality $q_j = q_k$ is trivially true or false, we can suppose that there is at most one equality between terms of type Q and that it is of the form $q = q_j$.
- If $t_{i,\varphi}, t'_{i,\varphi}$ have type Σ then they are of the form $c^{(j)}(\xi_1(\xi_2(\ldots))(\text{pos}^{(j)}))$ where the ξ_k's are $\mathit{Succ}^{(j)}$ or $\mathit{Pred}^{(j)}$. The equalities between terms of type Σ are all comparisons of letters among the values of $c^{(1)}(-k), \ldots, c^{(1)}(k), \ldots, c^{(n)}(-k), \ldots, c^{(n)}(k)$ where k is defined above.

This shows that the values of Φ depend solely on the value of q and those of the $c^{(j)}(\text{pos}^{(j)} + i)$'s for $j = 1, \ldots, n$ and $i = -k, \ldots, k$). This is exactly to say that what matters is the current state and the current letters in the n windows of diameter $2k+1$ centered at the positions of the n heads. Otherwise said, the tuple of arguments of the functional Φ is literally identical to the current values of the state plus the contents of the windows, that is a tuple in $Q \times \Sigma^{n(2k+1)}$.

Let us look at the image of Φ which is given through finite families of tuples of terms given by the β_φ's. Since the only terms of type Q are q and the q_i's. Thus, Φ can leave the dynamic symbol q unchanged or modify it to any value. The same is valid for the dynamic symbol $\mathfrak{s}$ (using what is said above about the domain of Φ, this proves the non easy direction of point 4).
Terms of type Σ name the contents of some $c^{(j)}$ at positions which are at distance $\leq k$ of the position of the j-th head. Thus Φ can modify the values of the $c^{(j)}$ in the windows around the positions of the heads.
Terms of type $\mathbb{Z}^{(j)}$ name an integer at distance k of the position of the j-th head. Thus Φ can move any head left or right of at most k cells. This proves the non easy direction of point 5. Thus, an EMA in $\mathcal{C}_T^{(n)}$ is literally identical to some window Turing machine. The converse is proved in a similar (much easier) way. ∎

Remark 3.5. A slight variation in the EMA model can have strong effect. For instance, suppose we add a constant 0 to the static signature and interpret it as 0 in the structure $\mathcal{M}_{\text{sta}}$. Then we get window Turing machines in which the head can jump to cell 0.

4. Random access machines

In order to *identify* RAMs with a simple class of EMAs, we introduce a slight variant of RAMs, which we call "transition RAM" (TRAM): 1) a bounded number of registers can be modified in one step, 2) it can test for equality to 0 and equality between combinations (via the fixed set of operations on $\mathbb{N}$) of the contents of the addressed registers, 3) the program is replaced by a transition function. Though this function operates on an infinite domain, it is finitarily defined via ground terms.

Definition 4.1 (n-transition RAMs)**.** Let $f_1, \ldots, f_p$ operations on non negative integers, A n-transition RAM (n-TRAM) with operations $f_1, \ldots, f_p$ is a tuple

$$(n, k, Q = \{q_0, \ldots, q_{r-1}\}, F^+, F^-, \delta, \rho_i, \tau_{i,j})_{i=1,\ldots,n,\ j=1,\ldots,k}$$

where

- n is the number of distinguished registers,
- Σ and Q are finite sets (the alphabet and the set of states),

- $F^+, F^- \subseteq Q \times \texttt{Bool}^p$ (accepting/rejecting final local configurations),
- $\delta : Q \times \texttt{Bool}^p \to Q$ (state transition),
- $\rho_i : Q \times \texttt{Bool}^p \to T$ (modification of register i) for $i = 1, \ldots, n$, where T is a finite family of terms built with the operations $f_1, \ldots, f_p$ and $n(1+k)$ constants (representing the contents of the addressed registers),
- $\tau_{i,j} : Q \times \texttt{Bool}^p \to T$ (modification of the register addressed through an iteration of j successive addressing, starting with register i), for $i = 1, \ldots, n$, $j = 1, \ldots, k$.

At any time the n-TRAM accesses registers $1, \ldots, n$ and the registers addressed addressed through at most k iterated addressing by these registers. The $p = n(1+k)(1 + \frac{n(1+k)-1}{2})$ Boolean arguments in the δ, ρ_i, τ_i's test equalities or equalities to 0 of the contents of the $n(1+k)$ adressed registers. Map δ tells how the state is modified. Maps $\rho_i, \tau_{i,j}$'s tell how the contents of the accessed registers are modified.
The notions of run, halt, acceptance and rejection are defined in the usual way.

Definition 4.2 (The class of EMAs for TRAMS). Let $f_1, \ldots, f_p$ operations on non negative integers. We denote by $\mathcal{C}^{(n)}_{\mathrm{TRAM}}$ the class of EMAs $\mathcal{A}$ which satisfy the following conditions.

(1) $\mathcal{A}$ has 4 sorts and its multidomain is $\mathcal{D} = (\mathbb{N}, \mathbb{N}^{\mathrm{addr}}, Q, \mathfrak{S})$ where $\mathbb{N}^{\mathrm{addr}}$ is a copy of $\mathbb{N}$, Q is a finite set with r elements, and $\mathfrak{S} = \{\mathrm{go}, \mathrm{acc}, \mathrm{rej}\}$.

(2) The signature $\mathcal{S}_{\mathrm{sta}}$ (for the static framework) contains $n + r + 3$ constants: $1, \ldots, n$ of type $\mathbb{N}$, $q_0, \ldots, q_{r-1}$ of type Q, "go", "acc", "rej" of type $\mathfrak{S}$, and $n+1$ unary function symbols *cast* of type $\mathbb{N} \to N^{\mathrm{addr}}$, and, for each $i = 1, \ldots, n$, f_i of type $\mathbb{N}^{k_i} \to \mathbb{N}$. Their interpretations in $\mathcal{M}_{\mathrm{sta}}$ are as follows: i) f_i is interpreted as the given operation on $\mathbb{N}$, ii) the cast function is interpreted as the identity from $\mathbb{N}$ to its copy N^{addr}, iii) $1, \ldots, n$, the q_i's and "go", "acc", "rej" are interpreted in the obvious way.

(3) The signature $\mathcal{S}^{\mathrm{sta}}_{\mathrm{input}}$ is empty.

(4) The signature $\mathcal{S}_{\mathrm{dyn}}$ contains two constants $q, \mathfrak{s}$ of types Q and $\mathfrak{S}$. Their interpretations in $\mathcal{M}_{\mathrm{ini}}$ are q_0 and "go".

(5) The signature $\mathcal{S}^{\mathrm{dyn}}_{\mathrm{input}}$ contains one unary function c of type $\mathbb{N}^{\mathrm{addr}} \to \mathbb{N}$.

Thus, the EMAs in $\mathcal{C}^{(n)}_{\mathrm{TRAM}}$ are defined as those having particular signature, multidomain, static framework and initialization of some dynamic symbols with no condition on the functional Φ (other than its semialgebraicity).

Theorem 4.3 (EMA representation theorem for TRAMs).
Any n-TRAM is literally identical *to some EMA in the class $\mathcal{C}^{(n)}_{TRAM}$. Conversely, any EMA in $\mathcal{C}^{(n)}_{TRAM}$ is* literally identical *to some n-TRAM.*

Proof. Analogous to the proof of Theorem 3.4. ∎

5. Other models

Similar results can be proved with finite atomata, stack automata Schönhage machines.

Let us mention an interesting feature occurring in the EMA modelization of Schönhage Storage Modification Machines (SMM) which illustrates what has been said in §2.1 about operations with values in products of domains. The tape of an SMM is a dynamic graph which may grow or loose nodes. To manage the current set of nodes of this graph-tape, it is convenient to introduce the following items:

• Among the sets of the multi-domain $\mathcal{D}$, there is an infinite set X (where all nodes are taken) and the set $\mathfrak{P}_{\mathit{fin}}(X)$ of finite subsets of X. There is no structure on X nor on $\mathfrak{P}_{\mathit{fin}}(X)$.

• In the signature $\mathcal{S}_{\mathrm{dyn}}$, there is a constant symbol U of type $\mathfrak{P}_{\mathit{fin}}(X)$ (it tells which nodes are in the current graph-tape).

• In the signature $\mathcal{S}_{\mathrm{sta}}$, there is a function symbol *new* with type $\mathfrak{P}_{\mathit{fin}}(X) \to X \times \mathfrak{P}_{\mathit{fin}}(X)$. It is interpreted as a choice function $A \mapsto (a, A \cup \{a\})$ which picks in X a point outside A, i.e. such that $a \notin A$.

To add a new node to the graph tape, we apply *new* to U. The constraint that both components of *new* have to be fired simultaneously and on the same argument insures that when a new node is picked, it is automatically added to (the interpretation) of U with no condition on the functional Φ.

6. Uniformly bounded non determinism

Uniformly bounded non determinism allows at each step at most k choices where k is some fixed constant independent of the step. EMAs with ‘such non determinism are defined as are deterministic EMAs with the following modification: *replace the semialgebraic functional Φ by finitely many such functionals.* All litteral identity results mentioned in the previous sections extend easily to the non deterministic cases.

7. External non determinism

We now deal with a more powerful kind of non determinism: that given by external choices which may be done during the run. This is the action of Gurevich's “Choose” instruction. To deal with such an “external non determinism”, we enrich EMAs with a fifth signature: the “external dynamic” signature $\mathcal{S}_{\mathrm{ext}}$. We illustrate this notion with the example of Chomsky type 0 grammars.

Definition 7.1. A grammar is a finite set of rules $(u_i, v_i)_{i=1,\dots,n}$ where the u_i, v_i's are words in an alphabet Σ. The associated relation $R \subseteq \Sigma^\star \times \Sigma^\star$ is defined as follows: a pair (U, V) is in R if and only if there exists a finite sequence $U = U_0, \dots, U_k = V$ such that, for all $j < k$ there exists words P, S and some $i = 1, \dots, n$ such that $U_j = Pu_iS$ and $U_{j+1} = Pv_iS$.

Definition 7.2. We denote by $\mathcal{C}_{\mathrm{gra}}$ the class of non deterministic EMAs

$$\mathcal{A} = (3;\ \mathcal{S}_{\mathrm{sta}}, \mathcal{S}^{\mathrm{sta}}_{\mathrm{input}}, \mathcal{S}^{\mathrm{dyn}}_{\mathrm{input}}, \mathcal{S}_{\mathrm{dyn}}, \mathcal{S}_{\mathrm{ext}};\ \mathcal{D};\ \mathcal{M}_{\mathrm{sta}}, \mathcal{M}_{\mathrm{ini}};\ \Phi)$$

which satisfy the following conditions.

(1) $\mathcal{A}$ has 3 sorts and its multidomain is $\mathcal{D} = (\mathbb{N}, \Sigma^*, \mathfrak{S})$ where Σ is a finite set.

(2) The signature $\mathcal{S}_{\mathrm{sta}}$ (for the static framework) contains finitely many binary function symbols subst_i, $i = 1, \dots, n$ of type $\mathbb{N} \times \Sigma^* \to \Sigma^*$. There is some family $(u_i, v_i)_{i=1,\dots,n}$ of pairs of words such that the interpretation in $\mathcal{M}_{\mathrm{sta}}$ (the static framework) of subst_i is the function which acts on a pair (p, U) as follows: if U contains the factor u_i in position p then it is replaced by v_i, else U is not modified.

(3) The signatures $\mathcal{S}^{\mathrm{sta}}_{\mathrm{input}}$ and $\mathcal{S}_{\mathrm{dyn}}$ are empty.

(4) The signature $\mathcal{S}^{\rm dyn}_{\rm input}$ contains one constant w of type Σ^*.

(5) The signature $\mathcal{S}_{\rm ext}$ (the external dynamic environment) contains one constant $Choose$ of type $\mathbb{N}$. Its interpretation during the run is given as an external action: its value changes at each step.

Thus, the EMAs in $\mathcal{C}^{(n)}_{\rm gra}$ are defined as those having particular signature, multidomain, static framework and initialization of some dynamic symbols with no condition on the functional Φ (other than its semialgebraicity).

Using the fact that iteration of substitutions is also a substitution, one can prove :

Theorem 7.3. *Any grammar is* literally identical *to some EMA in the class $\mathcal{C}_{gra}$. Conversely, any EMA in $\mathcal{C}_{gra}$ is* literally identical *to some grammar.*

References

[1] Andreas Blass, Nachum Dershowitz and Yuri Gurevich. Exact exploration. *Microsoft TechReport MSR-TR-2009-99*, 2009.

[2] Egon Börger. Unifying View of Models of Computation and System Design Frameworks. *Annals of Pure and Applied Logic*, 133: 149-171, 2005.

[3] Giuseppe Del Castillo and Yuri Gurevich and Karl Stroetmann. Typed Abstract State Machines. *Unfinished manuscript*, 25 pages, 1998.

[4] Nachum Dershowitz and Yuri Gurevich. A natural axiomatization of computability and proof of Church's Thesis. *Bulletin. of Symbolic Logic*, 14(3):299–350, 2008.

[5] Serge Grigorieff and Pierre Valarcher. Evolving MultiAlgebras unify all usual sequential computation models. `http://lacl.univ-paris12.fr/valarcher/`.

[6] S. Dexter and P. Boyle and Y. Gurevich. Gurevich Abstract State Machines and Schönhage Storage Modification Machines. *JUCS*, 3(4): 279–303, 1997.

[7] Yuri Gurevich. Reconsidering Turing's Thesis: towards more realistic semantics of programs. *Technical Report CRL-TR-38-84, EEC Dept, Univ. Michigan*, 1984.

[8] Yuri Gurevich. A new Thesis. *Abstracts, American Math. Soc.*, 1985.

[9] Yuri Gurevich. *Logic and the Challenge of Computer Science*. Current Trends in Theoretical Computer Science, ed. Egon Börger, Computer Sc. Press. 1–57, 1988.

[10] Yuri Gurevich. Evolving Algebras: An Introductory Tutorial. *Bul. EATCS*, 43: 264–284, 1991. Reprinted in *Current Trends in Theoretical Computer Science, 1993*, 266–29, World Scientific, 1993.

[11] Yuri Gurevich. *May 1997 Draft of the ASM Guide*. Tech Report CSE-TR-336-97, EECS Dept, University of Michigan, 1997.

[12] Y Gurevich. The Sequential ASM Thesis. *Bul. EATCS*, 67: 93–124, 1999. Reprinted in *Current Trends in Theoretical Comp. Sc., 2001*, 363–392, World Scientific, 2001.

[13] Yuri Gurevich. Sequential Abstract State Machines capture Sequential Algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, July 2000.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 429-440
www.stacs-conf.org

COLLAPSING AND SEPARATING COMPLETENESS NOTIONS UNDER AVERAGE-CASE AND WORST-CASE HYPOTHESES

XIAOYANG GU[1] AND JOHN M. HITCHCOCK[2] AND A. PAVAN[3]

[1] LinkedIn Corporation

[2] Department of Computer Science, University of Wyoming

[3] Department of Computer Science, Iowa State University

Abstract. This paper presents the following results on sets that are complete for NP.

(i) If there is a problem in NP that requires $2^{n^{\Omega(1)}}$ time at almost all lengths, then every many-one NP-complete set is complete under length-increasing reductions that are computed by polynomial-size circuits.

(ii) If there is a problem in co-NP that cannot be solved by polynomial-size nondeterministic circuits, then every many-one complete set is complete under length-increasing reductions that are computed by polynomial-size circuits.

(iii) If there exist a one-way permutation that is secure against subexponential-size circuits and there is a hard tally language in NP∩co-NP, then there is a Turing complete language for NP that is not many-one complete.

Our first two results use worst-case hardness hypotheses whereas earlier work that showed similar results relied on average-case or almost-everywhere hardness assumptions. The use of average-case and worst-case hypotheses in the last result is unique as previous results obtaining the same consequence relied on almost-everywhere hardness results.

1. Introduction

It is widely believed that many important problems in NP such as satisfiability, clique, and discrete logarithm are exponentially hard to solve. Existence of such intractable problems has a bright side: research has shown that we can use this kind of intractability to our advantage to gain a better understanding of computational complexity, for derandomizing probabilistic computations, and for designing computationally-secure cryptographic primitives. For example, if there is a problem in EXP (such as any of the aforementioned problems) that has $2^{n^{\Omega(1)}}$-size worst-case circuit complexity (i.e., that for all sufficiently large n, no subexponential size circuit solves the problem correctly on all instances of size

Key words and phrases: computational complexity, NP-completeness.

Gu's research was supported in part by NSF grants 0652569 and 0728806.

Hitchcock's research was supported in part by NSF grants 0515313 and 0652601 and by an NWO travel grant. Part of this research was done while this author was on sabbatical at CWI.

Pavan's research was supported in part by NSF grants 0830479 and 0916797.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2462

n), then it can be used to construct pseudorandom generators. Using these pseudorandom generators, BPP problems can be solved in deterministic quasipolynomial time [23]. Similar average-case hardness assumptions on the discrete logarithm and factoring problems have important ramifications in cryptography. While these hardness assumptions have been widely used in cryptography and derandomization, more recently Agrawal [1] and Agrawal and Watanabe [2] showed that they are also useful for improving our understanding of NP-completeness. In this paper, we provide further applications of such hardness assumptions.

1.1. Length-Increasing Reductions

A language is NP-complete if every language in NP is *reducible* to it. While there are several ways to define the notion of reduction, the most common definition uses polynomial-time computable many-one functions. Many natural problems that arise in practice have been shown to be NP-complete using polynomial-time computable many-one reductions. However, it has been observed that all known NP-completeness results hold when we restrict the notion of reduction. For example, SAT is complete under polynomial-time reductions that are one-to-one and length-increasing. In fact, all known many-one complete problems for NP are complete under this type of reduction [9]. This raises the following question: are there languages that are complete under polynomial-time many-one reductions but not complete under polynomial-time, one-to-one, length-increasing reductions? Berman [8] showed that every many-one complete set for E is complete under one-to-one, length-increasing reductions. Thus for E, these two completeness notions coincide. A weaker result is known for NE. Ganesan and Homer [17] showed that all NE-complete sets are complete via one-to-one reductions that are exponentially honest.

For NP, until recently there had not been any progress on this question. Agrawal [1] showed that if one-way permutations exist, then all NP-complete sets are complete via one-to-one, length-increasing reductions that are computable by polynomial-size circuits. Hitchcock and Pavan [20] showed that NP-complete sets are complete under length-increasing P/poly reductions under the measure hypothesis on NP [26]. Recently Buhrman et al. improved the latter result to show that if the measure hypothesis holds, then all NP-complete sets are complete via length-increasing, P/-computable functions with $\log\log n$ bits of advice [10]. More recently, Agrawal and Watanabe [2] showed that if there exist regular one-way functions, then all NP-complete sets are complete via one-one, length-increasing, P/poly-computable reductions. All the hypotheses used in these works require the existence of an *almost-everywhere hard* language or an *average-case hard* language in NP.

In the first part of this paper, we consider hypotheses that only concern the *worst-case hardness* of languages in NP. Our first hypothesis concerns the deterministic time complexity of languages in NP. We show that if there is a language in NP for which every correct algorithm spends more than 2^{n^ϵ} time at almost all lengths, then NP-complete languages are complete via P/poly-computable, length-increasing reductions. The second hypothesis concerns nondeterministic circuit complexity of languages in co-NP. We show that if there is a language in co-NP that cannot be solved by nondeterministic polynomial-size circuits, then all NP-complete sets are complete via length-increasing P/poly-computable reductions. For more formal statements of the hypotheses, we refer the reader to Section 3. We stress that these hypotheses require only worst-case hardness. The worst-case hardness is of course required at every length, a technical condition that is necessary in order to build a reduction that works at every length rather than just infinitely often.

1.2. Turing Reductions versus Many-One Reductions

In the second part of the paper we study the completeness notion obtained by allowing a more general notion of reduction—Turing reduction. Informally, with Turing reductions an instance of a problem can be solved by asking polynomially many (adaptive) queries about the instances of the other problem. A language in NP is Turing complete if there is a polynomial-time Turing reduction to it from every other language in NP. Though many-one completeness is the most commonly used completeness notion, Turing completeness also plays an important role in complexity theory. Several properties of Turing complete sets are closely tied to the separation of complexity classes. For example, Turing complete sets for EXP are sparse if and only if EXP contains polynomial-size circuits. Moreover, to capture our intuition that a complete problem is easy, then the entire class is easy, Turing reductions seem to be the "correct" reductions to define completeness. In fact, the seminal paper of Cook [13] used Turing reductions to define completeness, though Levin [25] used many-one reductions.

This raises the question of whether there is a Turing complete language for NP that is not many-one complete. Ladner, Lynch and Selman [24] posed this question in 1975, thus making it one of the oldest problems in complexity theory. This question is completely resolved for exponential time classes such as EXP and NEXP [33, 12]. We know that for both these classes many-one completeness differs from Turing-completeness. However progress on the NP side has been very slow. Lutz and Mayordomo [27] were the first to provide evidence that Turing completeness differs from many-one completeness. They showed that if the measure hypothesis holds, then the completeness notions differ. Since then a few other weaker hypotheses have been used to achieve the separation of Turing completeness from many-one completeness [3, 30, 31, 21, 29].

All the hypotheses used in the above works are considered "strong" hypotheses as they require the existence of an *almost everywhere hard* language in NP. That is, there is a language L in NP and every algorithm that decides L takes exponential-time an *all but finitely many* strings. A drawback of these hypotheses is that we do not have any candidate languages in NP that are believed to be almost everywhere hard.

It has been open whether we can achieve the separation using more believable hypotheses that involve average-case hardness or worst-case hardness. None of the proof techniques used earlier seem to achieve this, as the they crucially depend on the almost everywhere hardness.

In this paper, for the first time, we achieve the separation between Turing completeness and many-one completeness using average-case and worst-case hardness hypotheses. We consider two hypotheses. The first hypothesis states that there exist 2^{n^ϵ}-secure one-way permutations and the second hypothesis states that there is a language in NEEE$\cap$coNEEE that can not be solved in triple exponential time with logarithmic advice, i.e, NEEE $\cap$ coNEEE $\not\subseteq$ EEE/log. We show that if both of these hypothesis are true, then there is a Turing complete language in NP that is not many-one complete.

The first hypothesis is an average-case hardness hypothesis and has been studied extensively in past. The second hypothesis is a worst-case hardness hypothesis. At first glance, this hypothesis may look a little esoteric, however, it is only used to obtain hard tally languages in NP$\cap$co-NP that are sufficiently sparse. Similar hypotheses involving double and triple exponential-time classes have been used earlier in the literature [7, 15, 19, 14].

We use length-increasing reductions as a tool to achieve the separation of Turing completeness from many-one completeness. We first show that if one-way permutations exist then NP-complete sets are complete via length-increasing, quasipolynomial-time computable reductions. We then show that if the second hypothesis holds, then there is a Turing complete language for NP that is not complete via quasi polynomial-time, length-increasing reductions. Combining these two results we obtain our separation result.

2. Preliminaries

In the paper, we use the binary alphabet $\Sigma = \{0, 1\}$. Given a language A, A_n denotes the characteristic sequence of A at length n. We also view A_n as a boolean function from Σ^n to Σ. For languages A and B, we say that $A =_{\mathrm{io}} B$, if $A_n = B_n$ for infinitely many n. For a complexity class $\mathcal{C}$, we say that $A \in {}_{\mathrm{io}}\mathcal{C}$ if there is a language $B \in \mathcal{C}$ such that $A =_{\mathrm{io}} B$.

For a boolean function $f : \Sigma^n \to \Sigma$, $CC(f)$ is the smallest number s such that there is circuit of size s that computes f. A function f is quasipolynomial time computable (QP-computable) if can be computed deterministically in time $O(2^{\log^{O(1)} n})$. We will use the triple exponential time class EEE = DTIME($2^{2^{2^{O(n)}}}$), and its nondeterministic counterpart NEEE.

A language L is in NP/poly if there is a polynomial-size circuit C and a polynomial p such that for every x, x is in L if and only if there is a y of length $p(|x|)$ such that $C(x, y) = 1$.

Our proofs make use a variety of results from approximable sets, instance compression, derandomization and hardness amplification. We mention the results that we need.

Definition 2.1. A language A is *$t(n)$-time 2-approximable* [6] if there is a function f computable in time $t(n)$ such that for all strings x and y, $f(x, y) \neq A(x)A(y)$.

A language A is *io-lengthwise t(n)-time 2-approximable* if there is a function f computable in time $t(n)$ such that for infinitely many n, for every pair of n-bit strings x and y, $f(x, y) \neq A(x)A(y)$.

Amir, Beigel, Gasarch [4] proved that every polynomial-time 2-approximable set is in P/poly. Their proof also implies the following extension for a superpolynomial function $t(n)$.

Theorem 2.2 ([4]). *If A is io-lengthwise $t(n)$-time 2-approximable, then for infinitely many n, $CC(A_n) \leq t^2(n)$.*

Given a language H' in co-NP, let H be $\{\langle x_1, \cdots, x_n\rangle \mid |x_1| = \cdots = |x_n| = n, x_i \in H'\}$. Observe that a n-tuple consisting of strings of length n can be encoded by a string of length n^2. From now we view a string of length n^2 as an n-tuple of strings of length n.

Theorem 2.3 ([16, 11]). *Let H and H' be defined as above. Suppose there is a language L, a polynomial-size circuit family $\{C_m\}$, and a polynomial p such that for infinitely many n, for every $x \in \Sigma^{n^2}$, x is in H if and only if there is a string y of length $p(n)$ such that $C(x, y)$ is in $L^{\leq n}$. Then H' is in* ${}_{\mathrm{io}}$NP*/poly.*

The proof of Theorem 2.3 is similar to the proofs in [16, 11]. The difference is rather than having a polynomial-time many-one reduction, here we have a NP/poly many-one reduction which works infinitely often. The nondeterminism and advice in the reduction can be absorbed into the final NP/poly decision algorithm. The NP/poly decision algorithm works infinitely often, corresponding to when the NP/poly reduction works.

Definition 2.4. A function $f : \{0,1\}^n \to \{0,1\}^m$ is *s-secure* if for every $\delta < 1$, every $t \leq \delta s$, and every circuit $C : \{0,1\}^n \to \{0,1\}^m$ of size t, $\Pr[C(x) = f(x)] \leq 2^{-m} + \delta$. A function $f : \{0,1\}^* \to \{0,1\}^*$ is $s(n)$*-secure* if it is $s(n)$-secure at all but finitely many length n.

Definition 2.5. An $s(n)$*-secure one-way permutation* is a polynomial-time computable bijection $\pi : \{0,1\}^* \to \{0,1\}^*$ such that $|\pi(x)| = |x|$ for all x and π^{-1} is $s(n)$-secure.

Under widely believed average-case hardness assumptions about the hardness of the RSA cryptosystem or the discrete logarithm problem, there is a secure one-way permutation [18].

Definition 2.6. A *pseudorandom generator (PRG) family* is a collection of functions $G = \{G_n : \{0,1\}^{m(n)} \to \{0,1\}^n\}$ such that G_n is uniformly computable in time $2^{O(m(n))}$ and for every circuit of C of size n,

$$\left| \Pr_{x\in\{0,1\}^n}[C(x) = 1] - \Pr_{y\in\{0,1\}^{m(n)}}[C(G_n(y)) = 1 \right| \leq \frac{1}{n}.$$

There are many results that show that the existence of hard functions in exponential time implies PRGs exist. We will use the following.

Theorem 2.7 ([28, 23])**.** *If there is a language A in* E *such that $CC(A_n) \geq 2^{n^\epsilon}$ for all sufficiently large n, then there exist a constant k and a PRG family $G = \{G_n : \{0,1\}^{\log^k n} \to \{0,1\}^n\}$.*

3. Length-Increasing Reductions

In this section we provide evidence that many-one complete sets for NP are complete via length-increasing reductions. We use the following hypotheses.

Hypothesis 1. There is a language L in NP and a constant $\epsilon > 0$ such that L is not in $_{\text{io}}$DTIME(2^{n^ϵ}).

Informally, this means that every algorithm that decides L takes more than 2^{n^ϵ}-time on at least one string at every length.

Hypothesis 2. There is a language L in co-NP such that L is not in $_{\text{io}}$NP/poly.

This means that every nondeterministic polynomial size circuit family that attempts to solve L is wrong on on at least one string at each length.

We will first consider the following variant of Hypothesis 1.

Hypothesis 3. There is a language L in NP and a constant $\epsilon > 0$ such that for all but finitely many n, $CC(L_n) > 2^{n^\epsilon}$.

We will first show that Hypothesis 3 holds, then NP-complete sets are complete via length-increasing reductions. Then we describe how to modify the proof to derive the same consequence under Hypothesis 1. We do this because the proof is much cleaner with Hypothesis 3. To use Hypothesis 1 we have to fix encodings of boolean formulas with certain properties.

3.1. If NP has Subexponentially Hard Languages

Theorem 3.1. *If there is a language L in* NP *and an $\epsilon > 0$ such that for all but finitely many n, $CC(L_n) > 2^{n^\epsilon}$, then all* NP*-complete sets are complete via length-increasing,* P/poly *reductions.*

Proof. Let A be a NP-complete set that is decidable in time 2^{n^k}. Let L be a language in NP that requires 2^{n^ϵ}-size circuits at every length. Since SAT is complete via polynomial-time, length-increasing reductions, it suffices to exhibit a length-increasing, P/poly-reduction from SAT to A.

Let $\delta = \frac{\epsilon}{2k}$. Consider the following intermediate language

$$S = \left\{ \langle x, y, z \rangle \;\middle|\; |x| = |z|, |y| = |x|^\delta, \texttt{MAJ}[L(x), \mathrm{SAT}(y), L(z)] = 1 \right\}.$$

Clearly S is in NP. Since A is NP-complete, there is a many-one reduction f from S to A. We will first show that at every length n there exist strings on which the reduction f must be honest. Let

$$T_n = \left\{ \langle x, z \rangle \in \{0,1\}^n \times \{0,1\}^n \;\middle|\; L(x) \neq L(z),\ \forall y \in \{0,1\}^{n^\delta}\ |f(\langle x, y, z \rangle)| > n^\delta \right\}$$

Lemma 3.2. *For all but finitely many n, $T_n \neq \varnothing$.*

Assuming that the above lemma holds, we complete the proof of the theorem. Given a length m, let $n = m^{1/\delta}$. Let $\langle x_n, z_n \rangle$ be the first tuple from T_n. Consider the following reduction from SAT to A: Given a string y of length m, the reduction outputs $f(\langle x_n, y, z_n \rangle)$. Given x_n and y_n as advice, this reduction can be computed in polynomial time. Since n is polynomial in m, this is a P/poly reduction.

By the definition of T_n, $L(x_n) \neq L(z_n)$. Thus $y \in \mathrm{SAT}$ if and only if $\langle x_n, y, z_n \rangle \in S$, and so y is in SAT if and only if $f(\langle x_n, y, z_n \rangle)$ is in A. Again, by the definition of T_n, for every y of length m, the length of $f(\langle x_n, y, z_n \rangle)$ is bigger than $n^\delta = m$. Thus there is a P/poly-computable, length-increasing reduction from SAT to A. This, together with the proof of Lemma 3.2 we provide next, complete the proof of Theorem 3.1. ■

Proof of Lemma 3.2. Suppose $T_n = \varnothing$ for infinitely many n. We will show that this yields a length-wise 2-approximable algorithm for L at infinitely many lengths. This enables us to contradict the hardness of L. Consider the following algorithm:

(1) Input x, z with $|x| = |z| = n$.
(2) Find a y of length n^δ such that $|f(\langle x, y, x \rangle)| \leq n^\delta$.
(3) If no such y is found, Output 10.
(4) If y is found, then solve the membership of $f(\langle x, y, z \rangle)$ in A. If $f(\langle x, y, z \rangle) \in A$, then output 00, else output 11.

We first bound the running time of the algorithm. Step 2 takes $O(2^{n^\delta})$ time. In Step 4, we decide the membership of $f(\langle x, y, z \rangle)$ in A. This step is reached only if the length of $f(\langle x, y, z \rangle)$ is at most n^δ. Thus the time taken to for this step is $(2^{n^\delta})^k \leq 2^{n^{\epsilon/2}}$ time. Thus the total time taken by the algorithm is bounded by $2^{n^\epsilon/2}$.

Consider a length n at which $T_n = \varnothing$. Let x and z be any strings at this length. Suppose for every y of length n^δ, the length of $f(\langle x, y, z \rangle)$ is at least n^δ. Then it must be the case that $L(x) = L(z)$, otherwise the tuple $\langle x, z \rangle$ belongs to T_n. Thus if the above algorithm fails to find y in Step 2, then $L(x)L(z) \neq 10$.

Suppose the algorithm succeeds in finding a y in Step 2. If $f(\langle x, y, z\rangle) \in A$, then at least one of x or z must belong to L. Thus $L(x)L(z) \neq 00$. Similarly, if $f(\langle x, y, z\rangle) \notin A$, then at least one of x or z does not belong to L, and so $L(x)L(z) \neq 11$.

Thus L is 2-approximable at length n. If there exist infinitely many lengths n, at which T_n is empty, then L is infinitely-often, length-wise, $2^{n^\epsilon/2}$-time approximable. By Theorem 2.2, L has circuits of size 2^{n^ϵ} at infinitely many lengths. ■

Now we will describe how to modify the proof if we assume that Hypothesis 1 holds. Let L be the hard language guaranteed by the hypothesis. We will work with 3-SAT. Fix an encoding of 3CNF formulas such that formulas with same numbers of variables can be encoded as strings of same length. Moreover, we require that the formulas $\phi(x_1, \cdots, x_n)$ and $\phi(b_1, \cdots, b_i, x_{i+1}, \cdots, x_n)$ can be encoded as strings of same length, where $b_i \in \{0, 1\}$. Fix a reduction f from L to 3-SAT such that all strings of length n are mapped to formulas with n^r variables, $r \geq 1$. Let 3-SAT$'$ = 3-SAT $\cap \cup_r \Sigma^{n^r}$. It follows that that if there is an algorithm that decides 3-SAT$'$ such that for infinitely many n the algorithm runs in 2^{n^ϵ} time on all formulas with n^r variables, then L is in ioDTIME(2^{n^ϵ}).

Now the proof proceeds exactly same as before except that we use 3-SAT$'$ instead of L, i.e, our intermediate language will be

$$\{\langle x, y, z\rangle \mid \mathtt{MAJ}[\text{3-SAT}'(x), \text{SAT}(y), \text{3-SAT}'(z)]\} = 1.$$

Consider the set T_n as before. It follows that if T_n is empty at infinitely many lengths, then for infinitely many n, 3-SAT$'$ is 2-approximable on formulas with n^r variables. Now we can use the disjunctive self-reducibility of 3-SAT$'$ to show that there is a an algorithm that solves 3-SAT$'$ and for infinitely many n, this algorithm runs in DTIME(2^{n^ϵ})-time on formulas with n^r variables. This contradicts the hardness of L. This gives the following theorem.

Theorem 3.3. *If there is a language in* NP *that is not in* ioDTIME(2^{n^ϵ})*, then all* NP*-complete sets are complete via length-increasing* P/poly *reductions.*

3.2. If co-NP is Hard for Nondeterministic Circuits

In this subsection we show that Hypothesis 2 also implies that all NP-complete sets are complete via length-increasing reductions.

Theorem 3.4. *If there is a language L in* co-NP *that is not in* ioNP*/poly, then* NP*-complete sets are complete via* P/poly*-computable, length-increasing reductions.*

Proof. We find it convenient to work with co-NP rather than NP. We will show that all co-NP-complete languages are complete via P/poly, length-increasing reductions.

Let H' be a language in co-NP that is not in ioNP/poly. Let H be

$$\{\langle x_1, \cdots, x_n\rangle \mid \forall 1 \leq i \leq n, [x_i \in H' \text{ and } |x_i| = n]\}.$$

Note that every n-tuple that may potentially belong to H can be encoded by a string of length n^2.

Let $S = 0H' \cup 1\overline{SAT}$. It is easy to show that S is in co-NP and S is not in ioNP*/poly*. Observe that S is co-NP-complete via length-increasing reductions. Let A be any co-NP-complete language. It suffices to exhibit a length-increasing reduction from S to A.

Consider the following intermediate language:

$$L = \{\langle x, y, z\rangle \mid |x| = |z| = |y|^2, \mathtt{MAJ}[x \in H, y \in S, z \in H] = 1\}.$$

Clearly the above language is in co-NP. Let f be a many-one reduction from L to $\overline{A}$. As before we will first show at every length n that there exits strings x and z such that for every y in S the length of $f(\langle x, y, z\rangle)$ is at least n.

Lemma 3.5. *For all but finitely many n, there exist two strings x_n and z_n of length n^2 with $H(x_n) \neq H(z_n)$ and for every $y \in S^n$, $|f(\langle x_n, y, z_n\rangle)| > n$.*

Proof. Suppose not. Then there exist infinitely many lengths n at which for every pair of strings (of length n^2) x and z with $H(x) \neq H(z)$, there exist a y of length n such that $|f(x, y, z)| \leq n$.

From this we obtain a NP/poly-reduction from H to A such that for infinitely many n, for every x of length n^2, $|f(x)| \leq n$. By Theorem 2.3, this implies that H' is in ${}_{\text{io}}$NP/poly. We now describe the reduction. Given n let z_n be a string (of length n^2) that is not in H.

(1) Input $x, |x| = n^2$. Advice: z_n.
(2) Guess a string y of length n.
(3) If $|f(\langle x, y, z_n\rangle)| > n$, the output $\perp$.
(4) Output $f(\langle x, y, z_n)$.

Suppose $x \in H$. Since $z_n \notin H$, there exists a string y of length n such that $y \in S$ and $|f(\langle x, y, z_n\rangle)| \leq n$. Consider a path that correctly guesses such a y. Since $z_n \notin H$, and $y \in S$, $\langle x, y, z_n\rangle \in L$. Thus $f(\langle x, y, z_n\rangle) \in A^{\leq n}$. Thus there exists at least one path on which the reduction outputs a string from $L \cap \Sigma^{\leq n}$. Now consider the case $x \notin H$. On any path, the reduction either outputs $\perp$ or outputs $f(\langle x, y, z_n\rangle)$. Since both z_n and x are not in H, $\langle x, y, z\rangle \notin L$. Thus $f(\langle x, y, z_n\rangle) \notin A$ for any y.

Thus there is a NP/poly many-one reduction from H to L such that for infinitely many n, the output of the reduction, on strings of length n^2, on any path is at most n. By Theorem 2.3, this places H' in ${}_{\text{io}}$NP/poly.

Thus for all but finitely many lengths n, there exist strings x_n and z_n of length n^2 with $H(x_n) \neq H(z_n)$ and for every $y \in S^n$, the length of $f(\langle x_n, y, z_n\rangle)$ is at least n. ∎

This suggests the following reduction h from S to A. The reduction will have x_n and z_n as advice. Given a string y of length n, the reductions outputs $f(\langle x_n, y, z_n\rangle)$. This reduction is clearly length-increasing and is length-increasing on every string from S. Thus we have the following lemma.

Lemma 3.6. *Consider the above reduction h from S to A, for all $y \in S$, $|h(y)| > |y|$.*

Now we show how to obtain a length-increasing reduction on all strings. We make the following crucial observation.

Observation 3.7. For all but finitely many n, there is a string y_n of length n such that $y_n \notin S$ and $|f(\langle x_n, y_n, z_n\rangle)| > n$.

Proof. Suppose not. This means that for infinitely many n, for every y from $\overline{S} \cap \Sigma^n$, the length of $f(\langle x_n, y, z_n\rangle)$ is less than n. Now consider the following algorithm that solves S. Given a string y of length n, compute $f(\langle x_n, y, z_n\rangle)$. If the length of $f(\langle x_n, y, z_n\rangle) > n$, then accept y else reject y.

The above algorithm can be implemented in P/poly given x_n and z_n as advice. If $y \in S$, then we know that that the length of $f(\langle x_n, y, z_n\rangle)$ is bigger than n, and so the above algorithm accepts. If $y \notin S$, then by our assumption, the length of $f(\langle x_n, y, z_n\rangle)$ is at most n. In this case the algorithm rejects y. This shows that S is in ${}_{\text{io}}\text{P/poly}$ which in turn implies that H' is in ${}_{\text{io}}\text{P/poly}$. This is a contradiction. ■

Now we are ready to describe our length increasing reduction from S to A. At length n, this reduction will have x_n, y_n and z_n as advice. Given a string y of length n, the reduction outputs $f(\langle x_n, y, z_n\rangle)$ if the length of $f(\langle x_n, y, z_n\rangle)$ is more than n. Else, the reduction outputs $f(\langle x_n, y_n, z_n\rangle)$.

Since $H(x_n) \neq H(z_n)$, $y \in S$ if and only if $f(\langle x_n, y, z_n\rangle) \in A$. Thus the reduction is correct when it outputs $f(\langle x_n, y, z_n\rangle)$. The reduction outputs $f(\langle x_n, y_n, z_n\rangle)$ only when the length of $f(\langle x_n, y, z_n\rangle)$ is at most n. We know that in this case $y \notin S$. Since $y_n \notin S$, $f(\langle x_n, y_n, z_n) \notin A$.

Thus we have a P/poly-computable, length-increasing from S to A. Thus all co-NP-complete languages are complete via P/poly, length-increasing reductions. This immediately implies that all NP-complete languages are complete via P/poly-computable, length-increasing reductions. ■

4. Separation of Completeness Notions

In this section we consider the question whether the Turing completeness differs from many-one completeness for NP under two plausible complexity-theoretic hypotheses:

(1) There exists a 2^{n^ϵ}-secure one-way permutation.
(2) $\text{NEEE} \cap \text{coNEEE} \not\subseteq \text{EEE}/\log$.

It turns out that the first hypothesis implies that every many-one complete language for NP is complete under a particular kind of length-increasing reduction, while the second hypothesis provides us with a specific Turing complete language that is not complete under the same kind of length-increasing reduction. Therefore, the two hypotheses together separate the notions of many-one and Turing completeness for NP as stated in the following theorem.

Theorem 4.1. *If both of the above hypotheses are true, there is is a language that is polynomial-time Turing complete for* NP *but not polynomial-time many-one complete for* NP*.*

Theorem 4.1 is immediate from Lemma 4.2 and Lemma 4.3 below.

Lemma 4.2. *Suppose 2^{n^ϵ}-secure one-way permutations exist. Then for every* NP*-complete language A and every $B \in$* NP*, there is a quasipolynomial-time computable, polynomial-bounded, length-increasing reduction reduction f from B to A.*

A function f is *polynomial-bounded* if there is a polynomial p such that the length of $f(x)$ is at most $p(|x|)$ for every x.

Lemma 4.3. *If* $\text{NEEE} \cap \text{coNEEE} \not\subseteq \text{EEE}/\log$*, then there is a polynomial-time Turing complete set for* NP *that is not many-one complete via quasipolynomial-time computable, polynomial-bounded, length-increasing reductions.*

The proof of Lemma 4.2 will appear in the full paper. The remainder of this section is devoted to proving Lemma 4.3. It is well known that any set A over Σ^* can be encoded as a tally set T_A such that A is worst-case hard if and only if T_A is worst-case hard. For our purposes, we need an average-case version of the this equivalence. Below we describe particular encoding of languages using tally sets that is helpful for us and prove the average-case equivalence.

Let $t_0 = 2$, $t_{i+1} = t_i^2$ for all $i \in \mathbb{N}$. Let $\mathcal{T} = \left\{0^{t_i} \mid i \in \mathbb{N}\right\}$. For each $l \in \mathbb{N}$, let $\mathcal{T}_l = \left\{0^{t_i} \mid 2^l - 1 \leq i \leq 2^{l+1} - 2\right\}$. Observe that $\mathcal{T} = \bigcup_{l=0}^{\infty} \mathcal{T}_l$. Given a set $A \subseteq \{0,1\}^*$, let $T_A = \left\{ 0^{2^{2^{r_x}}} \middle| x \in A \right\}$, where r_x is the rank index of x in the standard enumeration of $\{0,1\}^*$. It is easy to verify that for all $l \in \mathbb{N}$ and every x,

$$x \in A \cap \{0,1\}^l \iff 0^{t_{r_x}} \in T_A \cap \mathcal{T}_l. \tag{4.1}$$

Lemma 4.4. *Let A and T_A be as above. Suppose there is a quasipolynomial time algorithm $\mathcal{A}$ such that for every l, on an ϵ fraction of strings from $\mathcal{T}_l$, this algorithm correctly decides the membership in T_A, and on the rest of the strings the algorithm outputs "I do not know". There is a $2^{2^{2^{k(l+1)}}}$-time algorithm $\mathcal{A}'$ for some constant k that takes one bit of advice and correctly decides the membership in A on $\frac{1}{2} + \frac{\epsilon}{2}$ fraction of the strings at every length l.*

We know several results that establish worst-case to average-case connections for classes such as EXP and PSPACE [34, 5, 22, 23, 32]. The following lemma establishes a similar connection for triple exponential time classes, and can be proved using known techniques.

Lemma 4.5. *If* NEEE∩coNEEE $\not\subseteq$ EEE/log*, then there is language L in* NEEE∩coNEEE *such that no* EEE/log *algorithm can decide L, at infinitely many lengths n, on more than $\frac{1}{2} + \frac{1}{n}$ fraction of strings from $\{0,1\}^n$.*

Now we are ready to prove Lemma 4.3.

Proof of Lemma 4.3. By Lemma 4.5, there is a language $L \in (\text{NEEE} \cap \text{coNEEE}) - \text{EEE}/\log$ such that no EEE/log algorithm can decide L correctly on more than a $\frac{1}{2} + \frac{1}{n}$ fraction of the inputs for infinitely many lengths n.

Without loss of generality, we can assume that $L \in \text{NTIME}(2^{2^{2^n}}) \cap \text{coNTIME}(2^{2^{2^n}})$ Let

$$T_L = \left\{ 0^{2^{2^{r_x}}} \middle| x \in L \right\}.$$

Clearly, $T_L \in \text{NP} \cap \text{coNP}$.

Define $\tau : \mathbb{N} \to \mathbb{N}$ such that $\tau(n) = \max\{i \mid t_i \leq n\}$. Now we will define our Turing complete language. Let

$$\text{SAT}_0 = \left\{0x \mid 0^{t_{\tau(|x|)}} \notin T_L \text{ and } x \in \text{SAT}\right\},$$

$$\text{SAT}_1 = \left\{1x \mid 0^{t_{\tau(|x|)}} \in T_L \text{ and } x \in \text{SAT}\right\}.$$

Let $A = \text{SAT}_0 \cup \text{SAT}_1$. Since L is in NP ∩ co-NP, A is in NP. The following is a Turing reduction from SAT to A: Given a formula x, ask queries $0x$ and $1x$, and accept if and only if at least one them is in A. Thus A is polynomial-time 2-tt complete for NP.

Suppose A is complete via length-increasing, polynomial-bounded, quasipolynomial-time reductions. Then there is such a reduction f from $\{0\}^*$ to A. There is a constant d such that f is n^d-bounded and runs in quasipolynomial time.

The following observation is easy to prove.

Observation 4.6. Let $y \in \{0,1\}^*$ and $b \in \{0,1\}$ be such that $f(0^{t_i}) = by$. Then $0^{t_{\tau(|y|)}} \in T_L$ if and only if $b = 1$.

Fix a length l. We will describe a quasipolynomial-time algorithm that will decide the membership in T_L on at least $\frac{1}{\log d}$ fraction of strings from $\mathcal{T}_l$, and says "I do not know" on other strings. By the Lemma 4.4, this implies that there is EEE/1 algorithm that decides L on more than $\frac{1}{2} + \frac{1}{2 \log d}$ fraction of strings from $\{0,1\}^l$. This contradicts the hardness of L and completes the proof.

Let $s = 2^l - 1$ and $r = 2^{l+1} - 2$. Recall that $\mathcal{T}_l = \left\{0^{t_i} \mid s \leq i \leq r\right\}$. Divide $\mathcal{T}_l$ in sets $T_0, T_2, \cdots T_r$ where $T_k = \left\{0^{t_i} \mid s + k \log d \leq r + (k+1) \log d\right\}$. This gives at least $\frac{2^l}{\log d}$ sets. Consider the following algorithm that decides T_L on strings from $\mathcal{T}_l$: Let 0^{t_j} be the input. Say, it lies in the set T_k. Compute $f(0^{t_{s+k \log d}}) = by$. If $t_{\tau(|y|)} \neq t_j$, then output "I do not know". Otherwise, accept 0^{t_j} if and only if $b = 1$. By Observation 4.6 this algorithm never errs. Since f is computable in quasipolynomial time, this algorithm runs in quasipolynomial time. Finally, observe that $t_{\tau(|y|)}$ lies between $t_{s+k \log d}$ and $t_{s+(k+1) \log d}$. Thus for every k, $0 \leq k \leq r$, there is at least one string from from T_k on which the above algorithm correctly decides T_L. Thus the above algorithm correctly decides T_L on at least $\frac{1}{\log d}$ fraction of strings from $\mathcal{T}_l$, and never errs. ∎

References

[1] M. Agrawal. Pseudo-random generators and structure of complete degrees. In *Proceedings of the Seventeenth Annual IEEE Conference on Computational Complexity*, pages 139–147, 2002.

[2] M. Agrawal and O. Watanabe. One-way functions and the isomorphism conjecture. Technical Report TR09-019, Electronic Colloquium on Computational Complexity, 2009.

[3] K. Ambos-Spies and L. Bentzien. Separating NP-completeness notions under strong hypotheses. *Journal of Computer and System Sciences*, 61(3):335–361, 2000.

[4] A. Amir, R. Beigel, and W. Gasarch. Some connections between bounded query classes and non-uniform complexity. *Information and Computation*, 186:104–139, 2003.

[5] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[6] R. Beigel. *Query-limited reducibilities*. PhD thesis, Stanford University, 1987.

[7] R. Beigel and J. Feigenbaum. On being incoherent without being very hard. *Computational Complexity*, 2(1):1–17, 1992.

[8] L. Berman. *Polynomial Reducibilities and Complete Sets*. PhD thesis, Cornell University, 1977.

[9] L. Berman and J. Hartmanis. On isomorphism and density of NP and other complete sets. *SIAM Journal on Computing*, 6:305–322, 1977.

[10] H. Buhrman, B. Hescott, S. Homer, and L. Torenvliet. Non-uniform reductions. *Theory of Computing Systems*. To appear.

[11] H. Buhrman and J. M. Hitchcock. NP-hard sets are exponentially dense unless NP $\subseteq$ coNP/poly. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity*, pages 1–7. IEEE Computer Society, 2008.

[12] H. Buhrman, S. Homer, and L. Torenvliet. Completeness for nondeterministic complexity classes. *Mathematical Systems Theory*, 24:179–200, 1991.

[13] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[14] J. Feigenbaum, L. Fortnow, S. Laplante, and A. Naik. On coherence, random-self-reducibility, and self-correction. *Computational Complexity*, 7:174–191, 1998.

[15] J. Feigenbaum, L. Fortnow, C. Lund, and D. Spielman. The powwer of adaptiveness and additional queries in random-self-reductions. *Computational Complexity*, 4:158–174, 1994.

[16] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 133–142, 2008.

[17] K. Ganesan and S. Homer. Complete problems and strong polynomial reducibilities. *SIAM J. Comput.*, 21(4):733–742, 1992.

[18] O. Goldreich, L. Levin, and N. Nisan. On constructing 1-1 one-way function. Technical Report TR95-029, ECCC, 1995.

[19] E. Hemaspaandra, A. Naik, M. Ogiwara, and A. Selman. P-selective sets and reducing search to decision vs. self-reducibility. *Journal of Computer and System Sciences*, 53(2):194–209, 1996.

[20] J. M. Hitchcock and A. Pavan. Comparing reductions to NP-complete sets. *Information and Computation*, 205(5):694–706, 2007.

[21] J. M. Hitchcock, A. Pavan, and N. V. Vinodchandran. Partial Bi-immunity, Scaled Dimension, and NP-Completeness. *Theory of Computing Systems*. To appear.

[22] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the 36th Annual Conference on Foundations of Computer Science*, pages 538–545, 1995.

[23] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Symposium on Theory of Computing*, pages 220–229, 1997.

[24] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

[25] L. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973. English translation of original in *Problemy Peredaci Informacii*.

[26] J. H. Lutz and E. Mayordomo. Measure, stochasticity, and the density of hard languages. *SIAM Journal on Computing*, 23(4):762–779, 1994.

[27] J. H. Lutz and E. Mayordomo. Cook versus Karp-Levin: Separating completeness notions if NP is not small. *Theoretical Computer Science*, 164(1–2):141–163, 1996.

[28] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[29] A. Pavan. Comparison of reductions and completeness notions. *SIGACT News*, 34(2):27–41, June 2003.

[30] A. Pavan and A. Selman. Separation of NP-completeness notions. *SIAM Journal on Computing*, 31(3):906–918, 2002.

[31] A. Pavan and A. Selman. Bi-immunity separates strong NP-completeness notions. *Information and Computation*, 188:116–126, 2004.

[32] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *JCSS: Journal of Computer and System Sciences*, 62, 2001.

[33] O. Watanabe. A comparison of polynomial time completeness notions. *Theoretical Computer Science*, 54:249–265, 1987.

[34] A. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 441-452
www.stacs-conf.org

REVISITING THE RICE THEOREM OF CELLULAR AUTOMATA

PIERRE GUILLON [1] AND GAÉTAN RICHARD [2]

[1] DIM - CMM, UMI CNRS 2807, Universidad de Chile, Av. Blanco Encalada 2120, 8370459 Santiago, Chile
E-mail address: pguillon@dim.uchile.cl

[2] Greyc, Université de Caen & CNRS, boulevard du Maréchal Juin, 14 000 Caen, France
E-mail address: grichard@info.unicaen.fr

Abstract. A cellular automaton is a parallel synchronous computing model, which consists in a juxtaposition of finite automata whose state evolves according to that of their neighbors. It induces a dynamical system on the set of configurations, *i.e.* the infinite sequences of cell states. The limit set of the cellular automaton is the set of configurations which can be reached arbitrarily late in the evolution. In this paper, we prove that all properties of limit sets of cellular automata with binary-state cells are undecidable, except surjectivity. This is a refinement of the classical "Rice Theorem" that Kari proved on cellular automata with arbitrary state sets.

Introduction

Among all results on undecidability, Rice's Theorem [Ric53] is probably one of the most important. It can be seen as stating the following: for any property on the functions computed by Turing machines, the set of corresponding machines is either trivial or undecidable. Following Church-Turing thesis, it is often thought that this result should remain true for other computational systems. It has, for instance, been extended with various restrictions to general dynamical systems [DB04], tilings [LW08] or, in a weaker form [CD04].

In this paper, we shall focus on a specific model known as cellular automata, introduced by Von Neumann [vN66]. Cellular automata are made of infinitely many cells endowed with a finite state and interacting locally and synchronously with each other. As this system does not have any way to give output, study of dynamics often uses the limit set, that consists of configurations which can appear arbitrarily late [Hur87, Čulík IIPY89]. In this domain, Jarkko Kari has already proved an equivalent of Rice theorem [Kar94b] for limit set. A similar, "perpendicular", result is also known for the trace, which consists on the evolution of only one fixed cell [CG07].

On the other hand, it is known that the property of being surjective (*i.e.* having a full limit set) is decidable and not trivial for one-dimensional cellular automata [AP72]. Such

1998 ACM Subject Classification: F.1.1 Models of Computation.
Key words and phrases: cellular automata, undecidability.
Thanks to the Projet Blanc ANR *Sycomore* and the Comité *ECOS-Sud*.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE
27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2474

a statement is not contradictory with Kari's theorem since it is not, properly speaking, a property of the limit set: a surjective CA can have the same limit set than a non-surjective one if the alphabets are distinct. Nevertheless, when fixing the alphabet, surjectivity becomes a property of the limit set. This leads to the question whether there exist other decidable properties on limit sets of cellular automata with fixed alphabet [Kar05, DFM00].

In this paper, we shall answer negatively to this question by extending the result of Kari when fixing the number of states, showing that all properties on limit sets other than surjectivity are either trivial or undecidable. Note that surjectivity is undecidable for higher dimensional cellular automata [Kar94a]. Our proofs use borders (example of similar constructions can be found in [DFV03, CFG07, Pou08]). The idea here is to restrict our study to nonsurjective cellular automata, since surjectivity is decidable. The (computable) forbidden words of the image can be used as border words.

The paper is organised as follows: first we give all the necessary definitions in Section 1 and some first properties of limit sets in Section 2. After that, we detail the core encoding of our construction in Section 3. With all this, we state the main Rice theorem in Section 4 before giving some concluding remarks in Section 5.

1. Definitions

We denote the set with two elements as $\mathbb{2} = \{0,1\}$. For any alphabet B, we denote as $B^{\mathbb{Z}}$ the set of *configurations* (all bi-infinite sequences over B). The length of some word $u \in B^*$ will be noted $|u|$. A *uniform* word or configuration is one where a single letter appears, with repetitions. For any configuration $x \in B^{\mathbb{Z}}$ or any word $x \in B^*$, $l, k \in \mathbb{Z}$, $x_{[\![l,k[\![}$ denotes the finite pattern $x_l x_{l+1} \dots x_{k-1}$. This notation is extended to the case where l or k is infinite.

A *cylinder* is the subset of configurations $[u]_i = \{x \in B^{\mathbb{Z}} \mid x_{[\![i,i+|u|[\![} = u\}$ sharing the common pattern $u \in B^*$ at position $i \in \mathbb{Z}$. Similarly, if $E \subset B^k$ for some $k \in \mathbb{N} \setminus \{0\}$, then $[E]_i$ will stand for the set of configurations $x \in B^{\mathbb{Z}}$ such that $x_{[\![i,i+k[\![} \in E$.

The set of configurations $B^{\mathbb{Z}}$ is a compact metric space when endowing it with the metric induced by the Cartesian product of the discrete topology on B. In this setting, open sets correspond to unions of cylinders.

If $b \in B$, then we note ${}^\infty b^\infty$ the configuration consisting in a uniform bi-infinite sequence of b. If $E \subset B^k$, with $k \in \mathbb{N} \setminus \{0\}$, then we will represent the set of bi-infinite sequences of words of E as ${}^\infty E^\infty = \left\{x \in B^{\mathbb{Z}} \middle| \exists i < k, \forall j \in \mathbb{Z}, x_{[\![i+jk,i+(j+1)k[\![} \in E\right\}$.

The following definition will be very helpful for future constructions: it allows to build borders so that some particular nonoverlapping zones of configurations can be recognized.

Definition 1.1. Let B be an alphabet and $n \in \mathbb{N} \setminus \{0\}$. A *strongly freezing word* $u \in B^n$ is a word such that for all $i \in [\![1,n[\![$, $uB^i \cap B^i u = \emptyset$. Equivalently, $[u]_0 \cap [u]_i = \emptyset$.
A set $E \subset B^n$ is *strongly freezing* if for all $i \in [\![1,n[\![$, $EB^i \cap B^i E = \emptyset$.

One first remark is that any word u can be extended to some strongly freezing word: simply take ub^k, where $b \neq u_0$ and $k \in \mathbb{N} \setminus \{0\}$ such that b^k does not appear in u.

The *shift* $\sigma : B^{\mathbb{Z}} \to B^{\mathbb{Z}}$ is defined for all $x \in B^{\mathbb{Z}}$ and $i \in \mathbb{Z}$ by $\sigma(x)_i = x_{i+1}$.

A *subshift* Σ is a closed subset of $B^{\mathbb{Z}}$ which is strongly invariant by shift, *i.e.* $\sigma(\Sigma) = \Sigma$. Equivalently, a subshift can be defined as the set of configurations avoiding a particular set $L \subset B^*$ of finite patterns, called *forbidden language*: $\left\{x \in B^{\mathbb{Z}} \middle| \forall i \in \mathbb{Z}, \forall u \in L, x_{[\![i,i+|u|[\![} \neq u\right\}$.

If the forbidden language L can be taken finite, then we say that Σ is *of finite type*; if it is empty it is the *full shift*. A subshift of finite type has *order* $k \in \mathbb{N} \setminus \{0\}$ if it admits a forbidden language $L \subset B^k$ containing only words of length k — or equivalently, of length at most k.

For any subshift Σ and $-\infty \leq l \leq m \leq +\infty$, denote $\mathcal{L}_{[\![l,m]\!]}(\Sigma) = \left\{ x_{[\![l,m]\!]} \middle| x \in \Sigma \right\}$. Note that, when $l - m$ is finite, it only depends on this difference, justifying the definition $\mathcal{L}_k(\Sigma) = \mathcal{L}_{[\![0,k[\![}(\Sigma)$ for $k \in \mathbb{N}$. We note $\mathcal{L}(\Sigma) = \bigcup_{k \in \mathbb{N}} \mathcal{L}_k(\Sigma) = \left\{ x_{[\![l,m]\!]} \middle| x \in \Sigma, l, m \in \mathbb{Z} \right\}$ the language of the subshift Σ.

Definition 1.2. A (one-dimensional) *cellular automaton* is a triplet (B, r, f) where B is a finite *alphabet* (or state set), $r \in \mathbb{N}$ is the neighborhood *radius* and $f : B^{2r+1} \to B$ is the *local transition function*.

A cellular automaton acts on elements of $B^{\mathbb{Z}}$ (called *configurations*) by synchronous and uniform application of the local transition function, inducing the *global transition function* $F : B^{\mathbb{Z}} \to B^{\mathbb{Z}}$, formally defined for all $x \in B^{\mathbb{Z}}$ and $i \in \mathbb{Z}$ by $F(x)_i = f(x_{i-r}, x_{i-r+1}, \ldots, x_{i+r})$. We will assimilate the cellular automaton with its global function.

It is easy to see that any cellular automaton commutes with the shift. In a more general way, Curtis, Hedlund and Lyndon proved that cellular automata correspond exactly to continuous self-maps of $B^{\mathbb{Z}}$ which commute whith the shift [Hed69].

Note that a local rule $f : B^{2r+1} \to B$ can be extended in $f : B^* \to B^*$ by $f(u) = f(u_{[\![0,2r+1[\![}) \ldots f(u_{[\![|u|-2r-1,|u|[\![})$.

A *partial cellular automaton* is the restriction of the global function of some cellular automaton to some subshift of finite type Σ. Note that it can be defined from an alphabet B, a radius $r \in \mathbb{N}$ and a local rule $f : \mathcal{L}_{2r+1}(\Sigma) \to B$.

For a cellular automaton (B, r, f), a state $b \in B$ is said to be *quiescent* if $f(b^{2r+1}) = b$. It is said to be *spreading* if $f(u) = b$ whenever the letter b appears in the word u.

Note that if F is a cellular automaton on alphabet B, then $F(B^{\mathbb{Z}})$ is a subshift. In particular, either F is surjective, or $F(B^{\mathbb{Z}})$ admits (at least) a forbidden pattern. It is easy to see that if $j \in \mathbb{N} \setminus \{0\}$, then F^j is also a cellular automaton, and the subshift $F^j(B^{\mathbb{Z}})$ is included in $F^{j-1}(B^{\mathbb{Z}})$.

The evolution being parallel and synchronous, we can see that the image of any uniform configuration remains uniform. The set of uniform configuration is then a finite subsystem, with an ultimate period $p \leq |B|$. In particular, F^p admits some quiescent state.

Definition 1.3. The *limit set* of a cellular automaton F is the set

$$\Omega_F = \bigcap_{j \in \mathbb{N}} F^j(B^{\mathbb{Z}})$$

of the configurations that can be reached arbitrarily late.

From the remark above, the limit set of the cellular automaton F always contains (at least) one uniform configuration. It is closed, and strongly invariant by F. More precisely, the restriction of F over Ω_F is its maximal surjective subsystem. In particular, F is surjective if and only if $\Omega_F = B^{\mathbb{Z}}$.

Moreover, it can be seen from the definition that $\Omega_F = \Omega_{F^k}$: the configurations that can be reached arbitrarily late are the same.

2. Preliminary results

In this section, we shall recall some classical results that will be needed in the proof.

The “Firing Squad” is a problem on algorithmics over cellular automata, introduced in 1964 by Moore and Myhill in [Moo64]: the goal is to synchronize cells of arbitrarily wide zones so that they all take the same given state at the same time. It led to different solutions (see [Maz96]); when dealing with infinitely many cells, we obtain that it is possible to make them get this state arbitrarily late in time, and Kari’s theorem was the first extrinsic purpose to this construction; we will reuse it as it was claimed.

Proposition 2.1 ([Kar94b])**.** *There exist some cellular automaton S on some alphabet B and some states $\kappa, \gamma \in B$, with κ spreading, such that:*

(1) *For any $J \in \mathbb{N}$, there is some configuration $z \in B^{\mathbb{Z}}$ such that $F^J(z) = {}^{\infty}\gamma^{\infty}$ and $\forall i \in \mathbb{Z}, j < J, F^j(z)_i \neq \gamma$;*
(2) *$\Omega_S \cap [\gamma]_0 \subset \{\kappa, \gamma\}^{\mathbb{Z}}$.*

To prove the undecidability of some property, we need to reduce to it some other property which is already known to be undecidable. It is classical to reduce the nilpotency problem, which was proved undecidable in [Kar92]. This proof reduced some tiling problem to the nilpotency, but actually, the CA involved all admitted some spreading state. Hence the following stronger result can be derived directly.

Theorem 2.2. *The problem*

Instance: *a cellular automaton N with some spreading state θ.*
Question: *is N nilpotent?*

is undecidable.

The restriction to cellular automata with spreading state is very convenient to allow constructions of products of cellular automata, thanks to the following result (see for instance [CG07] for a simple proof).

Proposition 2.3. *A cellular automaton N on some alphabet A with some spreading state $\theta \in A$ is nilpotent if and only if $\forall x \in A^{\mathbb{Z}}, \exists i \in \mathbb{Z}, j \in \mathbb{N}, N^j(x)_i = \theta$.*

3. Binary simulation

The main construction in Kari’s proof is based on a simultaneous simulation of several cellular automata thanks to some complex alphabet. In order to keep a fixed alphabet, we now need to encode additionnal information into binary configurations. This can be done thanks to the fact that one of the cellular automata is assumed to be non-surjective. The non-reachable portions of configurations can be used for the complex encodings.

Lemma 3.1. *Let C be an alphabet, Σ a subshift on alphabet $\mathbb{2}$ distinct from $\mathbb{2}^{\mathbb{Z}}$. Then we can build some strongly freezing language $E \subset \mathbb{2}^k \setminus \mathcal{L}_k(\Sigma)$, with $k \in \mathbb{N} \setminus \{0\}$, and some bijection $\xi : \mathcal{L}_k(\Sigma) \times C \to E$.*

Proof. The basic idea is to use the space outside Σ to compress the word of $\mathcal{L}(\Sigma)$ and make space for the additional information $v \in C$. However, to construct it freezing, we shall compress only the second half on the word. Let u be a forbidden pattern for Σ. Should we extend it as stated before, we can suppose that it is strongly freezing. Should we rename the letters, we can suppose that $C \subset \mathbb{2}^l$, with $l \in \mathbb{N} \setminus \{0\}$. As a subset of $(\mathbb{2}^{|u|} \setminus \{u\})^m$, $\mathcal{L}_{m|u|}(\Sigma)$ has cardinal less than $(2^{|u|}-1)^m$, and admits thus a bijection $\tilde{\xi}$ from $\mathcal{L}_{m|u|}(\Sigma)$ onto some subset of $\mathbb{2}^n$ whenever $2^n \geq (2^{|u|}-1)^m$. Let us take $m \geq \frac{2|u|+l}{|u|-\log(2^{|u|}-1)}$ and $n = m|u| - 2\,|u| - l$. We now take $k = 2m|u|$ and define $\xi : (z,v) \mapsto u z_{[\![0,m|u|[\![}\tilde{\xi}(z_{[\![m|u|,k[\![})vu$ (see Fig. 1) and $E = \xi(\mathcal{L}_k(\Sigma) \times C)$.

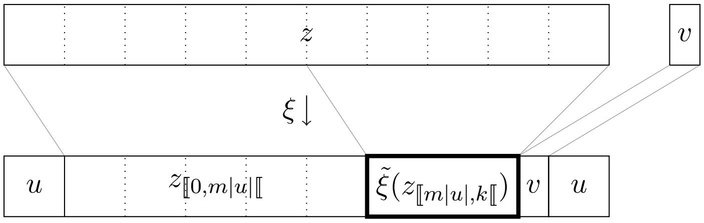


Figure 1: Encoding into strongly freezing alphabet

Now let $w \in E\mathbb{2}^i \cap \mathbb{2}^i E$ with $1 \leq i < k$. Note that $w_{[\![i,i+|u|[\![} = u$. But we also have $u = w_{[\![0,|u|[\![} = w_{[\![k-|u|,k[\![}$ and u is strongly freezing, so $|u| \leq i < k - 2\,|u|$. Moreover, $w_{[\![|u|,(m+1)|u|[\![}$ is in $\mathcal{L}_{m|u|}(\Sigma)$ and therefore does not contain the pattern u. Hence $i > m\,|u|$. Similarly, $w_{[\![i+|u|,i+(m+1)|u|[\![}$ cannot contain the pattern u, so $k - |u| \notin [\![i+|u|\,, i+m\,|u|[\![$, *i.e.* $i > k - 2\,|u|$, which gives a contradiction. ■

The language E will then be used as a particular alphabet, over which we can build configurations in $E^{\mathbb{Z}}$. This full shift can be more or less seen – up to a short initial shift – as the system $({}^\infty E^\infty, \sigma^k)$, but must not be confused with the subshift $({}^\infty E^\infty, \sigma)$ over $\mathbb{2}$. The key point in that construction is that the inclusion of the information of another shift can be done by a constant-space simulation: Σ and $C^{\mathbb{Z}}$ are, in an independent way, factors of respectively $({}^\infty E^\infty, \sigma)$ and of $E^{\mathbb{Z}}$ – or, thanks to freezingness, of $({}^\infty E^\infty, \sigma^k)$. This could not be done in the absence of any forbidden word u.

Given some partial cellular automaton G on some subshift of finite type Σ, some cellular automata N and S on alphabets $A \ni \theta$ and $B \ni \gamma, \kappa$. Considering $C = A \times B$, we can build E, k, ξ as in Lemma 3.1. As a local rule of radius 1, and with a little abuse of notation corresponding to commuting the products, ξ can be extended to injections $\xi : \mathcal{L}_{ik}(\Sigma) \times A^i \times B^i \to E^i$. Let δ_G, δ_N and δ_S be the local rules of G, N and S, and assume, without loss of generality, that S and N have same radius r_S and that G has radius $r_G < r_S k$. Define some cellular automaton $\Delta_{G,N,S}$ of radius $r = (r_S+1)k - 1$ and local rule $\delta : \mathbb{2}^{2r+1} \to \mathbb{2}$ defined as:

$$\delta(y) = \left| \begin{array}{lll} \delta_G(y_{\llbracket r-r_G, r+r_G\rrbracket}) & \text{if } y \in \mathcal{L}_{(2r+1)}(\Sigma) & (1) \\ z_i & \text{if } \left\{ \begin{array}{l} y \in \mathbb{2}^i E^{r_S} \xi(z,v,\gamma) E^{r_S} \mathbb{2}^{k-1-i} \\ 0 \le i < k, z \in \mathcal{L}_k(\Sigma), v \in A \end{array} \right. & (2) \\ \xi(z, \delta_N(v), \delta_S(w))_i & \text{if } \left\{ \begin{array}{l} y \in \mathbb{2}^i \xi(z,v,w) \mathbb{2}^{k-1-i} \\ 0 \le i < k, z \in \mathcal{L}_{(2r_S+1)k}(\Sigma) \\ v \in A^{r_S} \times A \setminus \{\theta\} \times A^{r_S} \\ w \in B^{r_S} \times B \setminus \{\gamma, \kappa\} \times B^{r_S} \end{array} \right. & (3) \\ 0 & \text{otherwise} & (4) \end{array} \right.$$

This rule is well-defined since the freezingness of E imposes the unicity of i in the cases (2) and (3). Intuitively, the constructed automaton behaves as G on Σ (1) and uses the freezing alphabet to simulate both automata N and S while keeping "compressed" an element of Σ (3). This element is uncompressed when automaton S reaches state γ (2). When N reaches state θ, S reaches state κ or when the encoding is invalid, the local transition goes to 0 (4).

Through the end of the section, the cellular automaton S will be a Firing Squad solution as built in Proposition 2.1. This will allow to make any configuration of Σ appear arbitrarily late during the evolution, since before the synchronization of S, the configuration of Σ will not be altered.

We will also assume that θ is a spreading state for the cellular automaton N. Intuitively, we wonder if N is nilpotent, and show that we can get an answer if we assume that some property over $\Delta_{G,N,S}$ is decidable.

Finally, we assume that the domain Σ of G is the subshift of finite type avoiding a single forbidden pattern u (of length less than k) such that $u_0 \neq 0 \neq u_{|u|-1}$. This last property allows that $0^*\mathcal{L}(\Sigma) \subset \mathcal{L}(\Sigma)$ and $\mathcal{L}(\Sigma)0^* \subset \mathcal{L}(\Sigma)$.

The following lemma shows that the non-encoding patterns will give words in Σ after one evolution step.

Lemma 3.2. *Let x be such that $\Delta_{G,N,S}(x) \in [u]_0$. Then there is some $i \in \rrbracket -k, 0\rrbracket$ such that $x \in [E^{2r_S+1}]_{i-r_Sk}$.*

Proof.

- If case (4) of the rule is applied to x in cell 0, then $\Delta_{G,N,S}(x)_0 = 0 \neq u_0$, hence $\Delta_{G,N,S}(x) \notin [u]_0$.
- If case (1) is applied to x in all cells of $\llbracket 0, k\llbracket$, then $\Delta_{G,N,S}(x)_{\llbracket 0,k\llbracket} \in [\mathcal{L}_k(\Sigma)]_0$, hence $\Delta_{G,N,S}(x) \notin [u]_0$.
- If x applies case (1) in cell 0, but there exists some $i \in \rrbracket 0, k\llbracket$ (say minimal) which applies some other rule. This means that the neighborhood $x_{\llbracket i-1-r, i-1+r\rrbracket}$ is in $\mathcal{L}_{2r+1}(\Sigma)$ whilst the neighborhood $x_{\llbracket i-r,i+r\rrbracket}$ is not, *i.e.* $x \in [u]_{i+r-|u|}$. As a result, all cells of $\llbracket i, k\llbracket$ (and many more) will see a non-homogeneous neighborhood and apply (4). $x_{\llbracket 0,k\llbracket} \in \mathcal{L}_i(\Sigma)0^{k-i} \subset \mathcal{L}_k(\Sigma)$, hence $\Delta_{G,N,S}(x) \notin [u]_0$.
- If x applies either (2) or (3) in cell 0, then we get the result. ■

The following lemma completes the previous one: not only cannot u appear from scratch, but no encoding pattern can appear from a non-encoding pattern.

Lemma 3.3. *Let x be such that $\Delta_{G,N,S}(x) \in [E]_0$. Then $x \in [E^{2r_S+1}]_{-r_S k}$.*

Proof.

- If case (3) of the rule is applied in some cell $j \in \llbracket 0,k\llbracket$, then $\Delta_{G,N,S}(x) \in [E]_{j-i}$ for some $j \in \llbracket 0,k\llbracket$. E being freezing, we get $\Delta_{G,N,S}(x) \in [E]_0$.
- Since all words of E contain some occurrence of u, Lemma 3.2 gives some $i \in \rrbracket -k, k-|u|\rrbracket$ such that $x_{\llbracket i-r_S k, i+(r_S+1)k\llbracket} = \xi(z,v,w)$, for some $z \in \mathcal{L}_{(2r_S+1)k}(\Sigma), v \in A^{2r_S+1}, w \in B^{2r_S+1}$. Assume that $i \geq 0$ – the symmetric case is similar. If the previous point does not occur, then case (2) is applied to cells of $\llbracket i, i+k\llbracket$, and either case (2) or case (3) to cells of $\llbracket i-k, i\llbracket$. Since $0^k\mathcal{L}_k(\Sigma) \subset \mathcal{L}_{2k}(\Sigma)$, both cases imply that $\Delta_{G,N,S}(x)_{\llbracket i-k,i+k\llbracket} \in \mathcal{L}_{2k}(\Sigma)$. This contradicts the assumption that $\Delta_{G,N,S}(x)_{\llbracket 0,k\llbracket} \in E$. ■

To study more in details the limit set of the constructed automaton, let us first consider the set

$$\Lambda = \bigcup_{\substack{0\leq i<k\\ -\infty\leq l<m\leq+\infty}} \left\{ x \in \mathbb{2}^{\mathbb{Z}} \middle| x_{\llbracket i+lk, i+mk\llbracket} \in E^{m-l} \text{ and } \forall j \notin \llbracket i+lk, i+mk\llbracket, x_j = 0 \right\}$$

of configurations or pieces of configurations of ${}^{\infty}E^{\infty}$ surrounded by 0. Note that this set does not depend on G – only on the subshift Σ. These partially encoding configurations correspond exactly to those of the limit set of our cellular automaton which are not in Σ, as proved below.

Lemma 3.4. $\Omega_{\Delta_{G,N,S}} \subset \Sigma \cup \Lambda$.

Proof. From Lemma 3.2, the image of the subshift which avoids all patterns of E is included in Σ, which itself is invariant. By shift-invariance, it is thus sufficient to prove that $\Omega_{\Delta_{G,N,S}} \cap [E]_0 \subset \Lambda$. One can remark that the patterns of $\left\{ v \in E\mathbb{2}^k\mathbb{2}^* \middle| v_{\llbracket k,2k\llbracket} \notin E \text{ and } v_{\llbracket k,2k\llbracket} \neq 0^k \right\}$ are forbidden in the image $\Delta_{G,N,S}(\mathbb{2}^{\mathbb{Z}})$. Indeed, if you apply case (3) of the rule in some cell and another one in another cell, then between these two cells there will be at least a range of k cells seeing a non-homogeneous neighborhood and applying case (4). By induction on $n \geq 1$, we can prove that the patterns of $\left\{ v \in E\mathbb{2}^{nk}\mathbb{2}^* \middle| v_{\llbracket k,2k\llbracket} \notin E \text{ and } v_{\llbracket k,2k+n-1\llbracket} \neq 0^{k+n-1} \right\}$ are forbidden in $\Delta^n_{G,N,S}(\mathbb{2}^{\mathbb{Z}})$, since at least the r_S extremal encoding patterns of E disappears at each step, whereas the non-zero patterns of Σ can spread only by $r_G < r_S k$ cells every step. In the limit, we obtain that all configurations of $\Omega_{\Delta_{G,N,S}}$ containing a pattern of E are in Λ. ■

In the case where N is nilpotent, we can see that the second part of the limit set is empty, and therefore we obtain the limit set of the original cellular automaton G.

Lemma 3.5. *If N is nilpotent, then $\Omega_{\Delta_{G,N,S}} = \Omega_G$.*

Proof. From Lemma 3.4 and the fact that $(\Delta_{G,N,S})_{|\Sigma} = G_{|\Sigma}$, it is sufficient to prove the emptyness of $\Omega_{\Delta_{G,N,S}} \cap \Lambda$. Let $x \in \Omega_{\Delta_{G,N,S}} \cap [E]_0$ and $J \in \mathbb{N}$. There exists $y^J \in \mathbb{2}^{\mathbb{Z}}$ such that $\Delta^J_{G,N,S}(y^J) = x$. Applying inductively Lemma 3.3, we obtain that $y^J_{\llbracket -Jr_S k,(Jr_S+1)k\llbracket} \in E^{2Jr_S+1}$. By compactness, there is some configuration y such that for any $i \in \mathbb{Z}$ and any $j \in \mathbb{N}$, $F^j(y)_{\llbracket ik,(i+1)k\llbracket} \in E$. Clearly, in the successive evolution step from y, case (3) of the local rule is always applied, which implies that there is a configuration in $A^{\mathbb{Z}}$ in the evolution of which θ never appears, hence contradicting the nilpotency of N. ■

When N is not nilpotent, we can see that there is a way to let the Firing Squad inject any configurations of Σ at any time, hiding the action of G: the limit set does not depend on G.

Lemma 3.6. *If N is not nilpotent, then $\Sigma \subset \Omega_{\Delta_{G,N,S}}$.*

Proof. Let $x \in \Sigma$. From Proposition 2.3, there is some configuration $y \in A^{\mathbb{Z}}$ such that for any $i \in \mathbb{Z}$ and any $j \in \mathbb{N}$, $N^j(x)_i \neq \theta$. Let $J \in \mathbb{N} \setminus \{0\}$. From Proposition 2.1, there is some configuration $z \in B^{\mathbb{Z}}$ such that $F^{J-1}(z) = {}^{\infty}\gamma^{\infty}$ and for any $j < J-1$, $F^j(z) \notin \{\gamma, \kappa\}$ (since κ is spreading). Consider now the configuration $\tilde{x}$ defined by $\forall i \in \mathbb{Z}, \tilde{x}_{[\![ik,(i+1)k[\![} = \xi(x_{[\![ik,(i+1)k[\![}, y_i, z_i)$. By a quick induction on $j < J$, we can see that for any cell $i \in \mathbb{Z}$, only case (3) of the local rule is used, and $\Delta_{G,N,S}(\tilde{x})_{[\![ik,(i+1)k[\![} = \xi(x_{[\![ik,(i+1)k[\![}, N^j(y)_i, S^j(z)_i)$. At time J, since $N^{J-1}(y)_i = \gamma$, the second part of the rule is applied and $\Delta_{G,N,S}(\tilde{x})_{[\![ik,(i+1)k[\![} = x_{[\![ik,(i+1)k[\![}$. As a result, $x \in \bigcap_{J \in \mathbb{N}\setminus\{0\}} \Delta^J_{G,N,S}(\mathbb{2}^{\mathbb{Z}})$. ■

4. Rice Theorem

The construction of the previous section allows us to separate the cases whether N is nilpotent in the same time as we separate properties of the limit set.

Lemma 4.1. *For any nontrivial property $\mathcal{P}$ over the limit sets of nonsurjective cellular automata on $\mathbb{2}$, there exist two cellular automata G_0, G_1 on alphabet $\mathbb{2}$ sharing the same quiescent state $q \in \mathbb{2}$, and such that $\Omega_{G_0} \in \mathcal{P}$, $\Omega_{G_1} \notin \mathcal{P}$.*

Proof. Take any nonsurjective cellular automaton M on $\mathbb{2}$ which has both 0 and 1 quiescent (such as a minimum cellular automaton). If its limit set satisfies $\mathcal{P}$, then take some nonsurjective cellular automaton G on $\mathbb{2}$ whose limit set does not satisfy $\mathcal{P}$. Then G^2 has the same limit set and some quiescent state $q \in \mathbb{2}$. If the limit set of M does not satisfy the property $\mathcal{P}$, then we can do the same with some nonsurjective cellular automaton G whose limit set does satisfy $\mathcal{P}$. ■

Lemma 4.2. *Let G_0, G_1 be two nonsurjective cellular automata on alphabet $\mathbb{2}$ sharing the same quiescent state $q \in \mathbb{2}$, and N a cellular automaton with a spreading state θ. Then we can build two cellular automata $F_i, i \in \{0,1\}$ such that $\Omega_{F_i} = \Omega_{G_i}$, $i \in \{0,1\}$, if N is nilpotent; $\Omega_{F_0} = \Omega_{F_1}$ otherwise.*

Proof. Should we invert 0 and 1 in the construction, we can assume that $q = 0$. Let u^i be a forbidden pattern of the (non-full) shift $G_i(\mathbb{2}^{\mathbb{Z}})$, and consider the word $u = 1u^0u^1 1$. The restrictions $\tilde{G}_i$ of G_i on the subshift Σ forbidding $\{u\}$ are partial cellular automata with the same limit sets than the respective G_i. Define $F_i = \Delta_{\tilde{G}_i,N,S}$ with S being as obtained in Proposition 2.1. Note that, except in the first case of the locale rule, the definitions of these two cellular automata are equivalent since they are based on the same subshift. If N is not nilpotent, then by Lemmas 3.4 and 3.6, $\Omega_{\Delta_{\tilde{G}_i,N,S}} = \Sigma \cup \Omega_{(\Delta_{\tilde{G}_i,N,S})_{|\Lambda}}$. It can be noted that the restrictions of $\Delta_{\tilde{G}_0,N,S}$ and $\Delta_{\tilde{G}_1,N,S}$ on Λ are equal. Hence $\Omega_{\Delta_{\tilde{G}_0,N,S}} = \Omega_{\Delta_{\tilde{G}_1,N,S}}$. Now if N is nilpotent, Lemma 3.5 gives the statement. ■

Here is now the main result.

Theorem 4.3. *Let $\mathcal{P}$ be a property satisfied by the limit set of at least one nonsurjective cellular automaton on $\mathbb{2}$, but not all. Then the problem*

Instance: *a cellular automaton F on $\mathbb{2}$.*
Question: $\Omega_F \in \mathcal{P}$*?*

is undecidable.

Proof. Assume such a property $\mathcal{P}$ is decidable. Let G_i, $i \in \{0,1\}$ be as in Lemma 4.1. Let us show a procedure to decide whether a given cellular automaton N on alphabet A with spreading state θ is nilpotent or not, which will contradict Theorem 2.2. We build the two cellular automata F_i as in Lemma 4.2 and we algorithmically check whether their limit sets satisfy property $\mathcal{P}$. If $\Omega_{F_0} \in \mathcal{P}$ and $\Omega_{F_1} \notin \mathcal{P}$, then N is nilpotent (otherwise the two limit sets would be equal). Otherwise, we know that one Ω_{F_i} is not equal to Ω_{G_i}, so N is not nilpotent. ■

From the decidability of the surjectivity problem, established in [AP72], we can rephrase the previous theorem as follows: surjectivity is the only nontrivial property of the limit sets of cellular automata on alphabet $\mathbb{2}$ to be decidable. Of course, this can be translated to any other fixed alphabet (of at least two letters).

5. Perspectives

This result is a very complete one, since it states that nothing can be said algorithmically with respect to how the long-time configurations look like. In spite of this, concrete examples of properties concerned are not so numerous, except nilpotency or apparition of a given state or pattern.

This is due to the fact that it does not include any dynamical idea. Various results have been obtained in this direction, about some properties of the restriction of the cellular automaton to the limit set [dLM09], the properties of the sequences of states taken by a particular cell [CG07], or the regularity of the languages obtained this way [dL06].

Among the properties that are not known to be concerned by our result, an important open problem consists in asking whether stability, which corresponds to the fact that the limit set is reached whithin a finite number of states (and the undecidability of which is not very hard to establish anyway), is a property of the limit sets or not. This issue is linked to the understanding of the different types of limit sets we can get with cellular automata, treated in particular in [Maa95].

Note that space-time diagrams of cellular automata, which represent the superposition of successive configurations in its application, are two-dimensional subshifts of finite type, *i.e.* drawings defined by some local constraints. Hence our result directly implies some kind of Rice theorem on subshift projections (multidimensional subshifts can be defined similarly).

Corollary 5.1. *Let $\mathcal{P}$ be a property satisfied by the limit set of at least one non-surjective cellular automaton on $\mathbb{2}$, but not all, and $\pi : \mathbb{2}^{\mathbb{Z}^2} \to \mathbb{2}^{\mathbb{Z}}$ defined by $\pi((x_{ij})_{i,j\in\mathbb{Z}}) = (x_{0j})_{j\in\mathbb{Z}}$. Then the problem*

Instance: *a subshift of finite type $\Sigma \subset \mathbb{2}^{\mathbb{Z}^2}$.*
Question: $\pi(\Sigma) \in \mathcal{P}$*?*

is undecidable.

The previous collary can obviously be generalized to any projection defined similarly from some k-dimensional tiling to some q-dimensional tiling, where $0 < q < k$. This does not include the property of being the full shift, but the undecidability of this property was already a consequence of the "perpendicular" Rice theorem in [CG07].

One may also wonder what happens for higher-dimensionnal cellular automata. In this case, our construction seems to extend well. Moreover, surjectivity is also undecidable which makes any non-trivial property undecidable.

Moreover, one can ask the same question on another characteristic set of cellular automata: the ultimate set, containing all the adhering values of orbits, studied for instance in [GR08]. One can notice that, when shifting enough a cellular automaton, the limit set is unchanged but the ultimate set becomes equal to the limit set. Hence, any nontrivial property of limit sets of cellular automata, except being a full shift, is an undecidable property of the ultimate set. We can wonder if it is the case for other nontrivial properties of ultimate sets.

More generally, the Firing Squad can be seen as a very powerful tool to touch the limit set. Our binary simulation can help hide its evolution within any alphabet. This could allow other complex constructions desolidarizing the simulation of a cellular automaton and the structure of its limit set. For instance, could we build an intrinsically universal cellular automaton (*i.e.* that can simulate any other cellular automaton) whose limit set is any given subshift of finite type?

References

[AP72] S. Amoroso and Y. N. Patt. Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. *Journal of Computer & System Sciences*, 6:448–464, 1972.

[CD04] Julien Cervelle and Bruno Durand. Tilings: recursivity and regularity. *Theoretical Computer Science*, 310(1–3):469–477, March 2004.

[CFG07] Julien Cervelle, Enrico Formenti, and Pierre Guillon. Sofic trace of a cellular automaton. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *Computation and Logic in the Real World, 3^{rd} Conference on Computability in Europe (CiE07)*, volume 4497 of *Lecture Notes in Computer Science*, pages 152–161, Siena, Italy, June 2007. Springer-Verlag.

[CG07] Julien Cervelle and Pierre Guillon. Towards a Rice theorem on traces of cellular automata. In Ludek Kučera and Antonín Kučera, editors, 32^{nd} *International Symposium on the Mathematical Foundations of Computer Science*, volume 4708 of *Lecture Notes in Computer Science*, pages 310–319, Český Krumlov, Czech Republic, August 2007. Springer-Verlag.

[Čulík IIPY89] Karel Čulík II, Jan K. Pachl, and Sheng Yu. On the limit sets of cellular automata. *SIAM Journal on Computing*, 18(4):831 – 842, 1989.

[DB04] Jean-Charles Delvenne and Vincent Blondel. Quasi-periodic configurations and undecidable dynamics for tilings, infinite words and Turing machines. *Theoretical Computer Science*, 319:127–143, 2004.

[DFM00] Marianne Delorme, Enrico Formenti, and Jacques Mazoyer. Open problems on cellular automata. Research Report 2000-25, École Normale Supérieure de Lyon, July 2000.

[DFV03] Bruno Durand, Enrico Formenti, and Georges Varouchas. On undecidability of equicontinuity classification for cellular automata. In Michel Morvan and Éric Rémila, editors, *Discrete Models for Complex Systems (DMCS'03)*, volume AB of *DMTCS Proc.*, pages 117–128. Discrete Mathematics and Theoretical Computer Science, June 2003.

[dL06] Pietro di Lena. Decidable properties for regular cellular automata. In Gonzalo Navarro, Leopoldo E. Bertossi, and Yoshiharu Kohayakawa, editors, 4^{th} *IFIP International Conference on Theoretical Computer Science (TCS'06)*, volume 209 of *International Federation for Information Processing*, pages 185–196, Santiago, Chile, August 2006. Springer.

[dLM09] Pietro di Lena and Luciano Margara. Undecidable properties of limit set dynamics of cellular automata. In Susanne Albers and Jean-Yves Marion, editors, 26^{th} *International Symposium on Theoretical Aspects of Computer Science (STACS'09)*, pages 337–348, Freiburg, Germany, February 2009. IBFI Schloss Dagstuhl.

[GR08] Pierre Guillon and Gaétan Richard. Nilpotency and limit sets of cellular automata. In Edward Ochmański and Jerzy Tyszkiewicz, editors, 33^{rd} *International Symposium on the Mathematical Foundations of Computer Science (MFCS'08)*, volume 5162 of *Lecture Notes in Computer Science*, pages 375–386, Toruń, Poland, August 2008. Springer-Verlag.

[Hed69] Gustav A. Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Mathematical Systems Theory*, 3:320 – 375, 1969.

[Hur87] Lyman P. Hurd. Formal language characterization of cellular automaton limit sets. *Complex systems*, 1:69 – 80, 1987.

[Kar92] Jarkko Kari. The nilpotency problem of one-dimensional cellular automata. *SIAM Journal on Computing*, 21(3):571–586, 1992.

[Kar94a] Jarkko Kari. Reversibility and surjectivity problems of cellular automata. *Journal of Computer and System Sciences*, 48(1):149–182, 1994.

[Kar94b] Jarkko Kari. Rice's theorem for the limit sets of cellular automata. *Theoretical Computer Science*, 127(2):229–254, 1994.

[Kar05] Jarkko Kari. Theory of cellular automata: A survey. *Theoretical Computer Science*, 334:3–33, 2005.

[LW08] Grégory Lafitte and Michael Weiss. Computability of tilings. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and Luke Ong, editors, 5^{th} *IFIP International Conference on Theoretical Computer Science (TCS'08)*, volume 273 of *International Federation for Information Processing*, pages 187–201, Milano, Italy, September 2008. Springer, Boston.

[Maa95] Alejandro Maass. On the sofic limit set of cellular automata. *Ergodic Theory & Dynamical Systems*, 15:663–684, 1995.

[Maz96] Jacques Mazoyer. On optimal solutions to the firing squad synchronization problem. *Theoretical Computer Science*, 168(2):367 – 404, 1996.

[Moo64] Edward F. Moore. The firing squad synchronisation problem. In Addison-Wesley, editor, *Sequential machines, Selected papers*, pages 213–214, 1964.

[Pou08] Victor Poupet. Translating partitioned cellular automata into classical cellular automata. In Bruno Durand, editor, *Journées Automates Cellulaires,* 1^{st} *Symposium on Cellular Automata (JAC'08)*, pages 130–140, Uzès, France, April 2008. MCCME Publishing House, Moscow.

[Ric53] Henry Grodon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[vN66] John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

ON OPTIMAL HEURISTIC RANDOMIZED SEMIDECISION PROCEDURES, WITH APPLICATION TO PROOF COMPLEXITY

EDWARD A. HIRSCH AND DMITRY ITSYKSON

Steklov Institute of Mathematics at St. Petersburg,
27 Fontanka, St.Petersburg, 191023, Russia
URL: http://logic.pdmi.ras.ru/~hirsch
URL: http://logic.pdmi.ras.ru/~dmitrits

Abstract. The existence of a (p-)optimal propositional proof system is a major open question in (proof) complexity; many people conjecture that such systems do not exist. Krajíček and Pudlák [KP89] show that this question is equivalent to the existence of an algorithm that is optimal[1] on all propositional tautologies. Monroe [Mon09] recently gave a conjecture implying that such algorithm does not exist.

We show that in the presence of errors such optimal algorithms *do* exist. The concept is motivated by the notion of heuristic algorithms. Namely, we allow the algorithm to claim a small number of false "theorems" (according to any polynomial-time samplable distribution on non-tautologies) and err with bounded probability on other inputs.

Our result can also be viewed as the existence of an optimal proof system in a class of proof systems obtained by generalizing automatizable proof systems.

1. Introduction

Given a specific problem, does there exist the "fastest" algorithm for it? Does there exist a proof system possessing the "shortest" proofs of the positive solutions to the problem? Although the first result in this direction was obtained by Levin [Lev73] in 1970s, these important questions are still open for most interesting languages, for example, the language of propositional tautologies.

1998 ACM Subject Classification: F.2.

Key words and phrases: propositional proof complexity, optimal algorithm.

Partially supported by grants RFBR 08-01-00640 and 09-01-12066, and the president of Russia grant "Leading Scientific Schools" NSh-4392.2008.1, by Federal Target Programme "Scientific and scientific-pedagogical personnel of the innovative Russia" 2009-2013 (contract N П265 from 23.07.2009). The second author is also supported by Russian Science Support Foundation.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2475

Classical version of the problem. According to Cook and Reckhow [CR79], a proof system is a polynomial-time mapping of all strings ("proofs") onto "theorems" (elements of certain language L; if L is the language of all propositional tautologies, the system is called a *propositional* proof system). The existence of a *polynomially bounded* propositional proof system (that is, a system that has a polynomial-size proof for every tautology) is equivalent to $\mathbf{NP} = \mathbf{co\text{-}NP}$. In the context of polynomial boundedness a proof system can be equivalently viewed as a function that given a formula and a "proof", verifies in polynomial time that a formula is a tautology: it must accept at least one "proof" for each tautology (*completeness*) and reject all proofs for non-tautologies (*soundness*).

One proof system Π_w is *simulated* by another one Π_s if the shortest proofs for every tautology in Π_s are at most polynomially longer than the shortest proofs in Π_w. The notion of *p-simulation* is similar, but requires also a polynomial-time computable function for translating the proofs from Π_w to Π_s. A *(p-)optimal* propositional proof system is one that (p-)simulates all other propositional proof systems.

The existence of an optimal (or p-optimal) propositional proof system is a major open question. If one would exist, it would allow to reduce the **NP** vs **co-NP** question to proving proof size bounds for just one proof system. It would also imply the existence of a complete disjoint **NP** pair [Raz94, Pud03]. Krajíček and Pudlák [KP89] show that the existence of a p-optimal system is equivalent to the existence of an algorithm that is optimal on all propositional tautologies, namely, it always solves the problem correctly and it takes for it at most polynomially longer to stop on every tautology than for any other correct algorithm *on the same tautology*. Monroe [Mon09] recently gave a conjecture implying that such algorithm does not exist. Note that Levin [Lev73] showed that an optimal algorithm does exist for finding witnesses to non-tautologies; however, (1) its behaviour on tautologies is not restricted; (2) after translating to the decision problem by self-reducibility the running time in the optimality condition is compared to the running time for *all shorter formulas as well*.

An *automatizable* proof system is one that has an automatization procedure that given a tautology, outputs its proof of length polynomially bounded by the length of the shortest proof in time bounded by a polynomial in the output length. The automatizability of a proof system Π implies polynomial separability of its canonical **NP** pair [Pud03], and the latter implies the automatizability of a system that p-simulates Π. This, however, does not imply the existence of (p-)optimal propositional proof systems in the class of automatizable proof systems. To the best of our knowledge, no such system is known to the date.

Proving propositional tautologies heuristically. An obvious obstacle to constructing an optimal proof system by enumeration is that no efficient procedure is known for enumerating the set of all complete and sound proof systems. Recently a number of papers overcome similar obstacles in other settings by considering either computations with non-uniform advice (see [FS06] for survey) or *heuristic* algorithms [FS04, Per07, Its09]. In particular, optimal propositional proof systems with advice do exist [CK07]. We try to follow the approach of heuristic computations to obtain a "heuristic" proof system. While our work is motivated by propositional proof complexity, i.e., proof systems for the set of propositional tautologies, our results apply to proof systems for any recursively enumerable language.

We introduce a notion of a *randomized heuristic automatizer* (a randomized semidecision procedure that may have false positives) and a corresponding notion of a *simulation*.

Its particular case, a deterministic automatizer (making no errors) for language L, along with deterministic simulations, can be viewed in two ways:

- as an automatizable proof system for L (note that such proof system can be identified with its automatization procedure; however, it may not be the case for randomized algorithms, whose running time may depend on the random coins), where simulations are p-simulations of proof systems;
- as an algorithm for L, where simulations are simulations of algorithms for L in the sense of [KP89].

Given $x \in L$, an automatizer must return 1 and stop. The question (handled by simulations) is how fast it does the job. For $x \notin L$, the running time does *not* matter. Given $x \notin L$, a deterministic automatizer simply must *not* return 1. A randomized heuristic automatizer may erroneously return 1; however, for "most" inputs it may do it only with bounded probability ("good" inputs). The precise notion of "most" inputs is: given an integer parameter d and a sampler for $\overline{L}$, "bad" inputs must have probability less than $1/d$ according to the sampler. The parameter d is handled by simulations in the way such that no automatizer can stop in time polynomial in d and the length of input unless an optimal automatizer can do that.

In Sect. 2 we give precise definitions. In Sect. 3 we construct an optimal randomized heuristic automatizer. In Sect. 4 we give a notion of heuristic probabilistic proof system and discuss the relation of automatizers to such proof systems.

2. Preliminaries

2.1. Distributional proving problems

In this paper we consider algorithms and proof systems *that allow small errors*, i.e., claim a small amount of wrong theorems. Formally, we have a probability distribution concentrated on non-theorems and require that the probability of sampling a non-theorem accepted by an algorithm or validated by the system is small.

Definition 2.1. We call a pair (D, L) a *distributional proving problem* if D is a collection of probability distributions D_n concentrated on $\overline{L} \cap \{0,1\}^n$.

In what follows we write $\Pr_{x \leftarrow D_n}$ to denote the probability taken over x from such distribution, while $\Pr_A$ denotes the probability taken over internal random coins used by algorithm A.

2.2. Automatizers

Definition 2.2. A (λ, ϵ)-*correct automatizer* for distributional proving problem (D, L) is a randomized algorithm A with two parameters $x \in \{0,1\}^*$ and $d \in \mathbb{N}$ that satisfies the following conditions:

(1) A either outputs 1 (denoted $A(\ldots) = 1$) or does not halt at all (denoted $A(\ldots) = \infty$);
(2) For every $x \in L$ and $d \in \mathbb{N}$, $A(x, d) = 1$.
(3) For every $n, d \in \mathbb{N}$,

$$\Pr_{r \leftarrow D_n} \left\{ \Pr_A \{A(r, d) = 1\} > \epsilon \right\} < \frac{1}{\lambda d}.$$

Here $\lambda > 0$ is a constant and $\epsilon > 0$ may depend on the first input (x) length. An *automatizer* is a $(1, \frac{1}{4})$-correct automatizer.

Remark 2.3. For recursively enumerable L, conditions 1 and 2 can be easily enforced at the cost of a slight overhead in time by running L's semidecision procedure in parallel.

In what follows, all automatizers are for the same problem (D, L).

Definition 2.4. The *time* spent by automatizer A on input (x, d) is defined as the median time

$$t_A(x,d) = \min\left\{t \in \mathbb{N} \;\middle|\; \Pr_A\{A(x,d) \text{ stops in time at most } t\} \geq \frac{1}{2}\right\}.$$

We will also use a similar notation for "probability p time":

$$t_A^{(p)}(x,d) = \min\left\{t \in \mathbb{N} \;\middle|\; \Pr_A\{A(x,d) \text{ stops in time at most } t\} \geq p\right\}.$$

Definition 2.5. Automatizer S simulates automatizer W if there are polynomials p and q such that for every $x \in L$ and $d \in \mathbb{N}$,

$$t_S(x,d) \leq \max_{d' \leq q(d \cdot |x|)} p(t_W(x,d') \cdot |x| \cdot d).$$

Definition 2.6. An *optimal* automatizer is one that simulates every other automatizer.

Definition 2.7. Automatizer A is *polynomially bounded* if there is a polynomial p such that for every $x \in L$ and every $d \in \mathbb{N}$,

$$t_A(x,d) \leq p(d \cdot |x|).$$

The following proposition follows directly from the definitions.

Proposition 2.8.

(1) *If W is polynomially bounded and is simulated by S, then S is polynomially bounded too.*

(2) *An optimal automatizer is not polynomially bounded if and only if no automatizer is polynomially bounded.*

3. Optimal automatizer

The optimal automatizer that we construct runs all automatizers in parallel and stops when the first of them stops (recall Levin's optimal algorithm for SAT [Lev73]). A major obstacle to this simple plan is the fact that it is unclear how to enumerate all automatizers efficiently (put another way, how to check whether a given algorithm is a correct automatizer). The plan of overcoming this obstacle (similar to constructing a complete public-key cryptosystem [HKN$^+$05] (see also [GHP09])) is as follows:

- Prove that w.l.o.g. a correct automatizer is very good: in particular, amplify its probability of success.
- Devise a "certification" procedure that distinguishes very good automatizers from incorrect automatizers with overwhelming probability.
- Run all automatizers in parallel, try to certify automatizers that stop, and halt when the first automatizer passes the check.

The amplification is obtained by repeating and the use of Chernoff bounds.

Proposition 3.1 (Chernoff bounds (see, e.g., [MR95, Chapter 4])). *Let $X_1, X_2, \ldots, X_n \in \{0,1\}$ be independent random variables. Then if X is the sum of X_i and if μ is $\mathbf{E}[X]$, for any δ, $0 < \delta \le 1$:*

$$\Pr\{X < (1-\delta)\mu\} < e^{-\mu\delta^2/2}, \qquad \Pr\{X > (1+\delta)\mu\} < e^{-\mu\delta^2/3}.$$

Corollary 3.2. *Let $X_1, X_2, \ldots, X_n \in \{0,1\}$ be independent random variables. Then if X is the sum of X_i and if $1 \ge \mu_1 \ge \mathbf{E}[X] \ge \mu_2 \ge 0$, for any δ, $0 < \delta \le 1$:*

$$\Pr\{X < (1-\delta)\mu_2\} < e^{-\mu_2\delta^2/2}, \qquad \Pr\{X > (1+\delta)\mu_1\} < e^{-\mu_1\delta^2/3}.$$

Lemma 3.3 (amplification). *Every automatizer W is simulated by a $(4, e^{-m/48})$-correct automatizer S, where $m \in \mathbb{N}$ may depend at most polynomially on $d \cdot |x|$ (for input (x,d)). Moreover, there are polynomials p and q such that for every $x \in L$ and $d \in \mathbb{N}$,*

$$t_S^{(1-e^{-m/64})}(x,d) \le \max_{d' \le q(d\cdot|x|)} p(t_W(x,d')). \tag{3.1}$$

Proof. $S(x,d)$ runs m copies of $W(x,4d)$ in parallel and stops as soon as the $\frac{3}{8}$ fraction of copies stop.

By Chernoff bounds, S is $(4, e^{-m/48})$-correct. The "strong" simulation condition (3.1) is satisfied because by Chernoff bounds the running time of the fastest $\frac{3}{8}$ fraction of executions is less than median time with probability at least $1 - e^{-m/64}$. ■

Theorem 3.4 (optimal automatizer). *Let (D, L) be a distributional proving problem, where L is recursively enumerable and D is polynomial-time samplable, i.e., there is a polynomial-time randomized Turing machine that given 1^n on input outputs x with probability $D_n(x)$ for every $x \in \{0,1\}^n$. Then there exists an optimal automatizer for (D, L).*

Proof. For algorithm A, we say that it is (λ,ϵ)-*correct for input length n and parameter d* if it it satisfies condition 3 of Definition 2.2 for n and d. If an algorithm is (λ,ϵ)-correct for every n (resp., every d), we omit n (resp., d).

In order to check an algorithm for correctness, we define a *certification* procedure that takes an algorithm A and distinguishes between the cases where A is $(4, \frac{1}{18d\log_*^2 n})$-correct for given n, d (from Lemma 3.3 we know that one can assume such correctness) or it is not $(1, \frac{1}{16d\log_*^2 n})$-correct ($(1, \frac{1}{16d\log_*^2 n})$-correct automatizers suffice for the correctness of further constructions). W.l.o.g. we may assume that

$$A \text{ satisfies conditions 1 and 2 of Definition 2.2} \tag{3.2}$$

(for the latter condition, notice that L is recursively enumerable and one may run its semidecision procedure in parallel).

The certification procedure has a subroutine Test that estimates the probability of A's error simply by repeating A and couting its faults.

```
TEST($A, x, d', T, l, f$):
  (1) Repeat for each $i \in \{1, \ldots, l\}$
      (a) If $A(x,d')$ stops in $T$ steps, let $c_i = 1$; otherwise let $c_i = 0$.
  (2) If $\sum_i c_i \ge l/f$, then reject; otherwise accept.
```

Lemma 3.5. *For every A, x, d', T, l, f,*

(1) *If $A(x, d')$ stops with probability less than $\frac{1}{1.01f}$, then* Test *will reject it with probability less than $e^{-\frac{l}{3.03\cdot 10^4\cdot f}}$.*

(2) *If $A(x, d')$ stops in time at most T with probability more than $\frac{1}{0.99f}$, then* Test *will accept it with probability less than $e^{-\frac{l}{2\cdot 10^4\cdot f}}$.*

Proof. Follows directly from Chernoff bounds. ■

```
Certify($A, n, d', T, k, l, f$):
  (1) Repeat for each $i \in \{1, \dots, k\}$
      (a) Generate $x_i$ according to $D_n$.
      (b) If Test($A, x_i, d', T, l, f$) rejects, let $b_i = 1$; otherwise let $b_i = 0$.
  (2) If $\sum_i b_i \geq k/(2d')$, then reject; otherwise accept.
```

Lemma 3.6. *Let $d, n, T \in \mathbb{N}$. Let A be an algorithm pretending to be an automatizer. Run*

$$\text{Certify}(A, n, d', T, k, l, f).$$

Then

(1) *If A is $(4, \frac{1}{1.011f})$-correct, then A is accepted by* Certify *almost for sure, failing with probability less than $e^{-\frac{k}{12d'}} + k \cdot e^{-\frac{l}{3.03\cdot 10^4\cdot f}}$.*

(2) *Let A^T be a restricted version of A that behaves similarly to A for T steps and enters an infinite loop afterwards. If A^T is not $(1, \frac{1}{0.99f})$-correct for length n and parameter d, then A is accepted by* Certify *with probability less than $e^{-\frac{k}{8d'}} + k \cdot e^{-\frac{l}{2\cdot 10^4\cdot f}}$.*

Proof. 1. Let $\Delta = \{x \in \operatorname{Im} D_n \mid \Pr\{A(x, d) = 1\} > \frac{1}{1.011f}\}$. By assumption, $D_n(\Delta) < \frac{1}{4d'}$.

The certification procedure takes k samples from D_n. For every sample $x_i \in \overline{L} \setminus \Delta$, the probability that the corresponding b_i equals 1 is less than $e^{-\frac{l}{3.03\cdot 10^4\cdot f}}$. Thus, the probability that there is a sample x_i from $\overline{L} \setminus \Delta$ that yields $b_i = 1$ is less than $k \cdot e^{-\frac{l}{3.03\cdot 10^4\cdot f}}$. Denote this unfortunate event by E. If it does not hold, only samples from Δ may cause $b_i = 1$ and by Chernoff's bound

$$\Pr\{\sum_i b_i \geq k/(2d') \mid \overline{E}\} < e^{-\frac{k}{12d'}}.$$

Thus, the total probability of reject is as claimed.

2. Let $\Delta = \{x \in \operatorname{Im} D_n \mid \Pr\{A(x, d) = 1\} > \frac{1}{0.99f}\}$. By assumption, $D_n(\Delta) \geq \frac{1}{d'}$.

The certification procedure takes k samples from D_n. For every sample $x_i \in \Delta$, the probability that the corresponding b_i equals 0 is less than $e^{-\frac{l}{2\cdot 10^4\cdot f}}$. Thus, the probability that there is a sample x_i from Δ that yields $b_i = 0$ is less than $k \cdot e^{-\frac{l}{2\cdot 10^4\cdot f}}$. Denote this unfortunate event by E. Assuming it does not hold only samples outside Δ may cause $b_i = 0$ and by Chernoff's bound

$$\Pr\{\sum_i b_i < k/(2d') \mid \overline{E}\} < e^{-\frac{k}{8d'}}.$$

■

We now define the optimal automatizer U. It works as follows:

```
U(x, d):
  (1) Let
        $n = |x|,$
        $d' = 16d\log_*^2 n,$
        $f = 17d\log_*^2 n,$
        $k = 12d' \ln(16d\log_*^2 n),$
        $l = (3.03 \cdot 10^4) \cdot f \cdot \ln(16kd\log_*^2 n).$
  (2) Run the following processes for $i \in \{1, \ldots, \log_* n\}$ in parallel:
      (a) Run $A_i(x, d')$, the algorithm with Turing number $i$ satisfying assumption (3.2),
          and compute the number of steps $T_i$ made by it before it stops.
      (b) If $\textsc{Certify}(A_i, n, d', T_i, k, l, f)$ accepts,
          then output 1 and stop $U$ (all processes).
  (3) If none of the processes has stopped, go into an infinite loop.
```

Correctness. We now show that U errs with probability less than $1/4$.

What are the inputs that cause U to error? For every such input x there exists $i \le \log_* n$ such that

$$u_x^i = \sum_{T=1}^{\infty} p_{x,T}^i c_T^i \ge \frac{1}{4\log_* n}, \tag{3.3}$$

where

$$\begin{aligned} p_{x,t}^i &= \Pr\{A_i(x, d') \text{ stops in exactly } t \text{ steps}\},\\ c_t^i &= \Pr\{\textsc{Certify}(A_i, n, d', t, k, l, f) \text{ accepts}\}. \end{aligned}$$

Let E_i be the set of inputs $x \notin L$ satisfying inequality (3.3).

We claim that $D(E_i) < \frac{1}{d\log_* n}$, which suffices to show the $(1, 1/4)$-correctness.

Assume the contrary. Let $T_i = \min\{t \mid c_t^i < e^{-\frac{k}{8d'}} + k \cdot e^{-\frac{l}{2\cdot 10^4 \cdot f}}\}$. Note that by Lemma 3.6 $A_i^{T_i - 1}$ is $(1, \frac{1}{0.99f})$-correct for n and d', i.e.,

$$\Pr_{x \leftarrow D_n}\{\sum_{T<T_*^i} p_{x,T}^i > \frac{1}{0.99f}\} < \frac{1}{d'}.$$

We omit i and n in the estimations that follow. Here is how we get a contradiction:

$$\frac{1}{4d\log_*^2 n} \le \frac{D(E_i)}{4\log_* n} = \sum_{x\in E_i} \frac{1}{4\log_* n} D(x) \le \sum_{x\in E_i} u_x D(x) \le$$

$$\sum_{x\notin L} u_x D(x) = \sum_{x\notin L} \sum_{T=1}^{\infty} p_{x,T} c_T D(x) =$$

$$\sum_{x\notin L} \left(\sum_{T<T_*} p_{x,T} c_T D(x) + \sum_{T\ge T_*} p_{x,T} c_T D(x) \right) \le$$

$$\sum_{T<T_*} \left(\sum_{x\notin L,\ \sum\limits_{t<T_*} p_{x,t} \le \frac{1}{0.99f}} p_{x,T} D(x) + \sum_{x\notin L,\ \sum\limits_{t<T_*} p_{x,t} > \frac{1}{0.99f}} p_{x,T} D(x) \right)$$

$$+ e^{-\frac{k}{8d'}} + k \cdot e^{-\frac{l}{2\cdot 10^4\cdot f}} \le$$

$$\frac{1}{0.99f} + \frac{1}{d'} + e^{-\frac{k}{8d'}} + k \cdot e^{-\frac{l}{2\cdot 10^4\cdot f}} < \frac{1}{16d\log_*^2 n} + \frac{1}{16d\log_*^2 n} + \frac{1}{8d\log_*^2 n} = \frac{1}{4d\log_*^2 n}.$$

Simulation. Assume we are give a correct automatizer A^s. Plug in $m = 48 \cdot \ln(18d\log_*^2 n)$ into Lemma 3.3. The lemma yields that A^s is "strongly" simulated by a $(4, \frac{1}{18d\log_*^2 n})$-correct automatizer A. It remains to estimate, for given "theorem" $x \in L$, the (median) running time of U in terms of $t_A^{(1-e^{-m/64})}(x,d) = t_A^{(1-\frac{1}{(18d\log_*^2 n)^{3/4}})}(x,d)$ (as we know that the latter is bounded by $\max\limits_{d'\le q(d\cdot|x|)} p(t_{A^s}(x,d'))$ for a polynomials p and q).

Since the definition of simulation is asymptotic, we consider only x of length greater than the Turing number of A. By Lemma 3.6, A is not certified with probability less than $e^{-\frac{k}{12d'}} + k \cdot e^{-\frac{l}{3.03\cdot 10^4\cdot f}} \le \frac{1}{8d\log_*^2 n}$. If A is certified, U stops in time upper bounded by a polynomial of the time spent by A with an overhead polynomial in $|x|$ and d for running other algorithms and the certification procedures. Thus the median time $t_U(x,d)$ is bounded by a polynomial in $|x|$, d, and $t_A^{(\frac{1}{2}+\frac{1}{8d\log_*^2 n})}(x,d) \le t_A^{(1-\frac{1}{(18d\log_*^2 n)^{3/4}})}(x,d)$. ■

4. Heuristic proof systems

In this section we define proof systems that make errors (claim a small fraction of wrong theorems). We consider automatizable systems of this kind and show that every such system defines an automatizer taking time at most polynomially larger than the length of the shortest proof in the initial system. This shows that automatizers form a more general notion than automatizable heuristic proof systems. The opposite direction is left as an open question.

Definition 4.1. Randomized Turing machine Π is a *heuristic proof system* for distributional proving problem (D, L) if it satisfies the following conditions.

(1) The running time of $\Pi(x, w, d)$ is bounded by a polynomial in d, $|x|$, and $|w|$.

(2) (Completeness) For every $x \in L$ and every $d \in \mathbb{N}$, there exists a string w such that $\Pr\{\Pi(x, w, d) = 1\} \geq \frac{1}{2}$. Every such string w is called a $\Pi^{(d)}$-proof of x.

(3) (Soundness) $\Pr_{x \leftarrow D_n}\{\exists w : \Pr\{\Pi(x, w, d) = 1\} > \frac{1}{4}\} < \frac{1}{d}$.

Definition 4.2. Heuristic proof system is *automatizable* if there is a randomized Turing machine A satisfying the following conditions.

(1) For every $x \in L$ and every $d \in \mathbb{N}$, with probability at least $\frac{1}{2}$ algorithm $A(x, d)$ outputs a correct $\Pi^{(d)}$-proof of size bounded by a polynomial in d, $|x|$, and $|w|$, where w is the shortest $\Pi^{(d)}$-proof of x.

(2) The running time of $A(x, d)$ is bounded by a polynomial in $|x|$, d, and the size of its own output.

Definition 4.3. We say that heuristic proof system Π_1 *simulates* heuristic proof system Π_2 if there exist polynomials p and q such that for every $x \in L$, the shortest $\Pi_1^{(d)}$-proof of x has size at most

$$p(d \cdot |x| \cdot \max_{d' \leq q(|x|d)} \{\text{the size of the shortest } \Pi_2^{(d')}\text{-proof of } x\}).$$

Note that this definition essentially ignores proof systems that have much shorter proofs for some inputs than the inputs themselves. We state it this way for its similarity to the automatizers case.

Definition 4.4. Heuristic proof system Π is *polynomially bounded* if there exists a polynomial p such that for every $x \in L$ and every $d \in \mathbb{N}$, the size of the shortest $\Pi^{(d)}$-proof of x is bounded by $p(|x|d)$.

Proposition 4.5. *If heuristic proof system* Π_1 *simulates system* Π_2 *and* Π_2 *is polynomially bounded, then* Π_1 *is also polynomially bounded.*

We now show how automatizers and automatizable heuristic proof systems are related.

Consider automatizable proof system (Π, A) for distributional proving problem (D, L) with recursively enumerable language L. Let us consider the following algorithm $A_\Pi(x, d)$:

```
(1) Execute 1000 copies of $A(x, d)$ in parallel.
    For each copy,
    (a) if it stops with result $w$, then
          • execute $\Pi(x, w, d)$ 10000 times;
          • if there were at least 4000 accepts of $\Pi$ (out of 10000), stop all parallel
            processes and output 1.
(2) Execute the enumeration algorithm for $L$; output 1 if this algorithm says that $x \in L$;
    go into an infinite loop otherwise.
```

Proposition 4.6. *If* (Π, A) *is a (correct) heuristic automatizable proof system for recursively enumerable language* L*, then* A_Π *is a correct automatizer for* $x \in L$ *and* $t_{A_\Pi}(x, d)$ *is bounded by polynomial in size of the shortest* Π_d*-proof of* x*.*

Proof. Soundness (condition 3 in Def. 2.2). Let $\Delta_n = \{x \in \overline{L} \mid \exists w : \Pr\{\Pi(x, w, d) = 1\} > \frac{1}{4}\}$. By definition, $D_n(\Delta_n) < \frac{1}{d}$. For $x \in \{0, 1\}^n \setminus \Delta_n$ and specific w, Chernoff bounds imply that $\Pi(x, w, d)$ accepts in 0.4 or more fraction of executions with exponentially small probability, which remains much smaller than $\frac{1}{4}$ even after multiplying by 1000.

Completeness (conditions 2 and 1 in Def. 2.2) is guaranteed by the execution of the semi-decision procedure for L.

Simulation. For $x \in L$, the probability that A errs 1000 times is negligible (at most 2^{-1000}). Thus with high probability at least one of the parallel executions of $A(x, d)$ outputs a correct Π_d-proof of size bounded by a polynomial in the size of the shortest Π_d-proof of x. For $x \in L$ and (correct) $\Pi^{(d)}$-proof w, Chernoff bounds imply that $\Pi(x, w, d)$ accepts in at least 0.4 fraction of executions with probability close to 1. Therefore, $t_{A_\Pi}(x, d)$ is bounded by a polynomial in $|x|$, d, and the size of the shortest Π_d-proof of x. ∎

5. Further research

One possible direction is to show that automatizers are equivalent to automatizable heuristic proof systems or, at least, that there is an optimal automatizable heuristic proof system. That may require some tweak in the definitions, because the first obstacle to proving the latter fact is the inability to check a candidate proof system for the non-existence of a much shorter (correct) proof than those output by a candidate automatizer.

Also Krajíček and Pudlák [KP89] and Messner [Mes99] list equivalent conditions for the existence of (deterministic) optimal and p-optimal proof systems. It seems promising (and, in some places, challenging) to prove similar statements in the heuristic setting.

Acknowledgements

During the work on the subject, we discussed it with many people. Our particular thanks go to (in the alphabetical order) Dima Antipov, Dima Grigoriev, and Sasha Smal.

References

[CK07] Stephen A. Cook and Jan Krajíček. Consequences of the provability of $NP \subseteq P/poly$. *The Journal of Symbolic Logic*, 72(4):1353–1371, 2007.

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, March 1979.

[FS04] Lance Fortnow and Rahul Santhanam. Hierarchy theorems for probabilistic polynomial time. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 316–324, 2004.

[FS06] Lance Fortnow and Rahul Santhanam. Recent work on hierarchies for semantic classes. *SIGACT News*, 37(3):36–54, 2006.

[GHP09] Dima Grigoriev, Edward A. Hirsch, and Konstantin Pervyshev. A complete public-key cryptosystem. *Groups, Complexity, Cryptology*, 1(1):1–12, 2009.

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In *Proc. of EUROCRYPT-2005*, 2005.

[Its09] Dmitry M. Itsykson. Structural complexity of AvgBPP. In *Proceedings of 4th International Computer Science Symposium in Russia*, volume 5675 of *Lecture Notes in Computer Science*, pages 155–166, 2009.

[KP89] Jan Krajíček and Pavel Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic*, 54(3):1063–1079, September 1989.

[Lev73] Leonid A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9:265–266, 1973.

[Mes99] Jochen Messner. On optimal algorithms and optimal proof systems. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 361–372, 1999.

[Mon09] Hunter Monroe. Speedup for natural problems and coNP?=NP. Technical Report 09-056, Electronic Colloquium on Computational Complexity, 2009.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

[Per07] Konstantin Pervyshev. On heuristic time hierarchies. In *Proceedings of the 22nd IEEE Conference on Computational Complexity*, pages 347–358, 2007.

[Pud03] Pavel Pudlák. On reducibility and symmetry of disjoint NP pairs. *Theoretical Computer Science*, 295(1–3):323–339, 2003.

[Raz94] Alexander A. Razborov. On provably disjoint NP-pairs. Technical Report 94-006, Electronic Colloquium on Computational Complexity, 1994.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 465-476
www.stacs-conf.org

WEAKENING ASSUMPTIONS FOR DETERMINISTIC SUBEXPONENTIAL TIME NON-SINGULAR MATRIX COMPLETION

MAURICE JANSEN

Institute for Theoretical Computer Science
FIT Building, Tsinghua University
100084 Beijing, China.
E-mail address: maurice.julien.jansen@gmail.com
URL: http://itcs.tsinghua.edu.cn/~mjjansen/

Abstract. Kabanets and Impagliazzo [9] show how to decide the circuit polynomial identity testing problem (CPIT) in deterministic subexponential time, assuming hardness of some explicit multilinear polynomial family $\{f_m\}_{m\geq 1}$ for arithmetic circuits.

In this paper, a special case of CPIT is considered, namely non-singular matrix completion (NSMC) under a low-individual-degree promise. For this subclass of problems it is shown how to obtain the same deterministic time bound, using a weaker assumption in terms of the *determinantal complexity* $\mathrm{dc}(f_m)$ of f_m.

Building on work by Agrawal [17], hardness-randomness tradeoffs will also be shown in the converse direction, in an effort to make progress on Valiant's VP versus VNP problem. To separate VP and VNP, it is known to be sufficient to prove that the determinantal complexity of the $m \times m$ permanent is $m^{\omega(\log m)}$. In this paper it is shown, for an appropriate notion of explicitness, that the existence of an explicit multilinear polynomial family $\{f_m\}_{m\geq 1}$ with $\mathrm{dc}(f_m) = m^{\omega(\log m)}$ is equivalent to the existence of an efficiently computable *generator* $\{G_n\}_{n\geq 1}$ *for* multilinear NSMC with seed length $O(n^{1/\sqrt{\log n}})$. The latter is a combinatorial object that provides an efficient deterministic black-box algorithm for NSMC. "Multilinear NSMC" indicates that G_n only has to work for matrices $M(x)$ of *poly*(n) size in n variables, for which $\det(M(x))$ is a multilinear polynomial.

1. Introduction

Let $\mathbb{F}$ be a field of characteristic zero, let $\mathbb{Q} \subseteq \mathbb{F}$ denote the field of rational numbers, and let $X_n = \{x_1, x_2, \ldots, x_n\}$ be a set of variables. $\mathcal{A}_{\mathbb{F}}(X_n)$ denotes the set of affine forms over X_n and $\mathbb{F}$. In this paper we study a special case of circuit polynomial identity testing, namely the non-singular matrix completion problem over $\mathbb{F}$. Matrix completion is an important problem, both in theory and in practice. The history of the problem dates back to work by Lovász [1] and Edmonds [2].

1998 ACM Subject Classification: F.2.3 Tradeoffs among Complexity Measures.

Key words and phrases: computational complexity, arithmetic circuits, hardness-randomness tradeoffs, identity testing, determinant versus permanent.

This work was supported in part by the National Natural Science Foundation of China Grant 60553001, and the National Basic Research Program of China Grant 2007CB807900,2007CB807901.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2477

As was done in [3] for CPIT, we study non-singular matrix completion under a promise restriction on individual degrees:

Problem. $\mathrm{NSMC}_r^k(\mathbb{F})$: $k \times k$ Non-Singular Matrix Completion over $\mathbb{F}$ with individual degrees at most r.

- Input: A $k \times k$ matrix $M(x)$ with entries in $\mathcal{A}_{\mathbb{F}}(X_n)$.
- Promise : Individual degrees of the polynomial $\det(M)$ are bounded by r.
- Question: Does there exist $a \in \mathbb{F}^n$ such that $\det M(a) \neq 0$?

Over a field of characteristic zero, the problem is equivalent to asking whether $\det M(x) \not\equiv 0$. Since $\det_n$ has $O(n^6)$ size skew circuits [4], and is universal for skew circuits (Implicit in [5], see Proposition 3.1), $\mathrm{NSMC}_{r(n)}^{poly(n)}(\mathbb{F})$ is equivalent to identity testing $poly(n)$ size skew circuits over $\mathbb{F}$, under the *semantic* promise that the circuit outputs a polynomial with individual degrees bounded by $r(n)$. Over $\mathbb{Q}$, for any $r(n)$, the latter can be verified with a coRP-algorithm, using the Schwartz-Zippel Lemma [6, 7]. Moreover, Lovász showed that a random assignment for x maximizes the rank of $M(x)$ with high probability [1].

Whether there exists an efficient deterministic algorithm for matrix completion is a major open problem. Currently, such an algorithm exists only for special instances. For example, Ivanyos, Karpinkski and Saxena give a polynomial time deterministic algorithm for finding a maximum rank completion, provided $M(x)$ is of the form $M_0 + x_1M_1 + x_2M_2 + \ldots + x_nM_n$, where $M_1, M_2, \ldots, M_n$ are rank one matrices [8].

Kabanets and Impagliazzo provide algebraic hardness-randomness tradeoffs for CPIT [9]. They show that the existence of an explicit polynomial with super-polynomial arithmetic circuit size, implies CPIT, and hence NSMC, can be decided deterministically in time 2^{n^ϵ}, for any $\epsilon > 0$, provided n is large enough. In order to make progress towards unconditionally proven deterministic subexponential time algorithms for NSMC, it is important to consider whether the same bound can be obtained for NSMC under any weaker assumptions.

In this paper we will only assume hardness of an explicit polynomial for skew circuits, or equivalently, we make hardness assumptions in terms of *determinantal complexity* [10]. In other words, we aim for specialized algebraic hardness-randomness tradeoffs for the skew circuit model. For this, we will use the hardness-randomness tradeoffs for constant-depth arithmetic circuits due to Dvir, Shpilka and Yehudayoff [3] as a starting point.

Another motivation is the VP versus VNP question, or the permanent versus determinant problem [10]. The latter problem asks us to prove lower bounds for the determinantal complexity of an explicit[1] polynomial. We firmly establish the role of NSMC in the quest for such lower bounds, firstly, by the characterization mentioned in the abstract. Secondly, it is shown that the existence of an explicit multilinear polynomial family $\{f_m\}_{m\geq 1}$ with $\mathrm{dc}(f_m) = m^{\omega(1)}$ is equivalent to the existence of an efficiently computable multilinear generator $\{G_n\}_{n\geq 1}$ *for* $\mathrm{NSMC}_1^{poly(n)}$ with seed length $\lceil n^\epsilon \rceil$, for some $0 < \epsilon < 1$.

2. Results

We require some formal definitions to properly state the results.

[1] Necessarily in the sense of Definition 2.2. A sufficient condition would require an even more stringent notion.

Definition 2.1 ([10])**.** The *determinantal complexity* $\mathrm{dc}(f)$ of a polynomial $f \in \mathbb{F}[X_n]$ is defined to be the minimum size of a matrix M with entries in $\mathcal{A}_{\mathbb{F}}(X_n)$ such that $\det M = f$.

We use standard definitions of arithmetic circuits and formulas with binary addition and multiplication operations (See [11]). Arithmetic circuit complexity of f is denoted by $L(f)$. A *skew* circuit satisfies that for every multiplication gate one of its inputs is a variable or a constant. $L_{skew}(f)$ denotes skew circuit size of f. The following is our notion of explicitness of a multilinear polynomial:

Definition 2.2. Let $\{f_m\}_{m\geq 1}$ be a family of multilinear polynomials with $f_m \in \mathbb{Z}[x_1, x_2, \ldots, x_m]$. We say this family is *explicit* provided there exists a deterministic Turing machine running in time $2^{O(m)}$, that on input $e \in \{0,1\}^m$, outputs the binary representation of the coefficient of the monomial $x_1^{e_1} x_2^{e_2} \ldots x_m^{e_m}$ of f_m.

Hardness Hypothesis 1 (HH1)**.** There exists an explicit family of multilinear polynomials $\{f_m\}_{m\geq 1}$, such that $L(f_m) = m^{\omega(1)}$.

Hardness Hypothesis 2 (HH2)**.** There exists an explicit family of multilinear polynomials $\{f_m\}_{m\geq 1}$, such that $\mathrm{dc}(f_m) = m^{\omega(1)}$.

If in the above we replace $m^{\omega(1)}$ by $m^{\omega(\log m)}$, we refer to this as Strengthened HH1 and Strengthened HH2.

Proposition 2.3. *HH2 is equivalent to the statement that there exists an explicit family of multilinear polynomials $\{f_m\}_{m\geq 1}$, such that $L_{skew}(f_m) = m^{\omega(1)}$. A similar statement holds for Strengthened HH2, but with $m^{\omega(1)}$ replaced by $m^{\omega(\log m)}$.*

Proof. In one direction this follows from the fact that the $n \times n$ determinant has skew circuits of size $O(n^6)$ [4]. For the converse, apply the fact that if f_m can be computed by a skew circuit of size s, then $\mathrm{dc}(f_m) = O(s)$ (Implicit in [5], see Proposition 3.1). ∎

Proposition 2.4. *Strengthened HH1 $\Rightarrow$ Strengthened HH2 $\Rightarrow$ HH1 $\Rightarrow$ HH2 .*

Proof. The first and the last implication follow from Proposition 2.3. To show that Strengthened HH2 $\Rightarrow$ HH1, suppose we have an explicit multilinear p-family $\{f_m\}_{m\geq 1}$, such that $\mathrm{dc}(f_m) = m^{\omega(\log m)}$. This implies $\mathrm{dc}(f_m) = m^{\omega(\log m)}$, even when restricting to $m \in \mathcal{M}$, for any infinite set $\mathcal{M}$. If $L(f_m) \notin m^{\omega(1)}$, then there exists constant $c > 0$ and an infinite set $\mathcal{M}'$, such that $L(f_m) \leq m^c$, for all $m \in \mathcal{M}'$. Using the construction of [12], we obtain formulas for f_m of size $2^{O(\log L(f_m) \log m)} = m^{O(\log m)}$, for $m \in \mathcal{M}'$. Hence by [5], $\mathrm{dc}(f_m) = m^{O(\log m)}$, for $m \in \mathcal{M}'$. This is a contradiction. ∎

Our algorithms will be of the black-box kind. This is formalized as follows:

Definition 2.5. For a function $\ell : \mathbb{N} \to \mathbb{N}$, a multilinear $(\ell(n), n)$-*generator* for $\mathrm{NSMC}_r^k(\mathbb{F})$ is given by a multilinear polynomial mapping $G_n : \mathbb{F}^{\ell(n)} \to \mathbb{F}^n$. We say G_n *provides a test for* $\mathrm{NSMC}_r^k(\mathbb{F})$, if for any instance $M(x)$ of $\mathrm{NSMC}_r^k(\mathbb{F})$, it holds that

$$(\exists a \in \mathbb{F}^n), \det M(a) \neq 0 \text{ iff } (\exists b \in \mathbb{F}^{\ell(n)}), \det M(G_n(b)) \neq 0.$$

Families $\{G_n\}_{n\geq 1}$ of generators are also simply called "generator". For a generator $\{G_n\}_{n\geq 1}$ with coefficients in $\mathbb{Z}$, we say it is *efficiently computable*, if there exists a deterministic Turing machine M that runs in time $2^{O(\ell(n))}$, so that on input (i, n, e), where i

and n are given in binary and $e \in \{0,1\}^{\ell(n)}$, M computes the binary representation of the coefficient of the monomial $x_1^{e_1}x_2^{e_2}\dots x_{\ell(n)}^{e_{\ell(n)}}$ of the ith component $(G_n)_i$ in the image of G_n.

We are now ready to state the results.

Theorem 2.6. *If HH2 holds over $\mathbb{F}$, then for any $0 < \epsilon < 1$, there exists an efficiently computable multilinear $(\lceil n^\epsilon \rceil, n)$-generator $\{G_n\}_{n\geq 1}$, such that for any $k(n) \in n^{O(1)}$ and $r(n) \in O(1)$, G_n provides a test for $\mathrm{NSMC}_{r(n)}^{k(n)}(\mathbb{F})$, for all large enough n.*

Theorem 2.7. *If Strengthened HH2 holds over $\mathbb{F}$, then there exists an efficiently computable multilinear $(O(n^{1/\sqrt{\log n}}), n)$-generator $\{G_n\}_{n\geq 1}$, such that for any $k(n) \in n^{O(1)}$ and $r(n) \in 2^{O(\sqrt{\log n})}$, G_n provides a test for $\mathrm{NSMC}_{r(n)}^{k(n)}(\mathbb{F})$, for all large enough n.*

From this we will derive the following:

Theorem 2.8. *If HH2 holds over $\mathbb{Q}$, then non-singular matrix completion over $\mathbb{Q}$ for matrices $M(x)$ of poly(n) size and with coefficients of poly(n) bits, where the individual degrees of $det(M(x))$ are bounded by a constant, can be decided deterministically in time 2^{n^ϵ}, for any $\epsilon > 0$, provided n is large enough.*

Theorem 2.9. *If Strengthened HH2 holds over $\mathbb{Q}$, then non-singular matrix completion over $\mathbb{Q}$ for matrices $M(x)$ of poly(n) size and with coefficients of poly(n) bits, can be decided deterministically in time $2^{O(n^{1/\sqrt{\log n}} \log n)}$, under the promise that individual degrees of $det(M(x))$ are bounded by $2^{O(\sqrt{\log n})}$.*

A central technical part of this paper is the following "Root Extraction Lemma" for skew circuits, which is of independent interest:

Lemma 2.10. *Let $n, s,$ and m be integers with $s \geq n$. Let $P(x,y) \in \mathbb{F}[X_n, y]$ be a non-zero polynomial such that $L_{skew}(P) = s$. Let $f \in \mathbb{F}[X_n]$ be a polynomial with $\deg(f) = m$ such that $P(x, f(x)) \equiv 0$. Then $L_{skew}(f) \leq s \cdot 2^{O(\log^2 m)} r^{4+\log m}$, where $r = \deg_y(P)$.*

Finally, we also prove the following randomness-to-hardness results:

Theorem 2.11. *If for some $0 < \epsilon < 1$, there exists an efficiently computable multilinear $(\lceil n^\epsilon \rceil, n)$-generator $\{G_n\}_{n\geq 1}$, such that for any $k(n) \in n^{O(1)}$, G_n provides a test for $\mathrm{NSMC}_1^{k(n)}(\mathbb{F})$, for all large enough n, then HH2 holds over $\mathbb{F}$.*

Theorem 2.12. *If there exists an efficiently computable multilinear $(O(n^{1/\sqrt{\log n}}), n)$-generator $\{G_n\}_{n\geq 1}$, such that for any $k(n) \in n^{O(1)}$, G_n provides a test for $\mathrm{NSMC}_1^{k(n)}(\mathbb{F})$, for all large enough n, then Strengthened HH2 holds over $\mathbb{F}$.*

Theorem 2.12 & Theorem 2.7 and Theorem 2.11 & Theorem 2.6 provide us with characterizations, which we summarize as follows:

Corollary 2.13.

(1) *HH2 holds over $\mathbb{F}$ if and only if there exists an efficiently computable multilinear $(\lceil n^\epsilon \rceil, n)$-generator $\{G_n\}_{n\geq 1}$, for some $0 < \epsilon < 1$, such that for all $k(n) \in n^{O(1)}$, G_n provides a test for $\mathrm{NSMC}_1^{k(n)}(\mathbb{F})$, for all large enough n.*

(2) *Strengthened HH2 holds over $\mathbb{F}$ if and only if there exists an efficiently computable multilinear $(O(n^{1/\sqrt{\log n}}), n)$-generator $\{G_n\}_{n\geq 1}$, such that for all $k(n) \in n^{O(1)}$, G_n provides a test for $\mathrm{NSMC}_1^{k(n)}(\mathbb{F})$, for all large enough n.*

3. Preliminaries

For a polynomial f, $H_k(f)$ denotes the homogeneous part of degree k, and $H_{\leq k}(f) \triangleq \sum_{i=0}^{k} H_i(f)$.

An *algebraic branching program* (ABP) Φ over $\mathbb{F} \cup X_n$ is given by a directed acyclic graph G with source node s and sink node t. Edges of G are labeled with elements of $X_n \cup \mathbb{F}$. The *weight* of a directed path in Φ is defined to be the product of the edge labels. The polynomial computed by Φ is defined to be the sum of weights over all directed s,t-paths. For the size of Φ we count the number of edges in G. For a polynomial f, $B(f)$ is the size of any smallest ABP computing f. This generalizes in the obvious way to multi-output ABPs, by having several sink nodes $t_1, t_2, \ldots, t_m$. One easily proves the following proposition:

Proposition 3.1. *$L_{skew}(f) = \Theta(B(f))$.*

We will use this to switch freely between skew circuits and ABPs. The latter model gives us some convenience. For example, for ABPs it is easy to see that if $f(x_1, x_2, \ldots, x_n)$ is computed by an ABP A of size s_A, and g is computed by an ABP B of size s_B, then $f(g, x_2, \ldots, x_n)$ can be computed by an ABP of size $O(s_A s_B)$. Indeed, simply replace each edge labeled with x_1 in A with the s,t-dag given by B. Addition and multiplication of ABPs is done by parallel and series composition, respectively.

Proposition 3.2. *Suppose Φ is a skew circuit of size s computing $f \in \mathbb{F}[X_n]$. Then for any i, there exists a skew circuit of size $O(s \cdot i)$ computing $H_j(f)$ for all $0 \leq j \leq i$.*

Proof. This is achieved using the standard homogenization trick of keeping for each gate in Φ, $i+1$ many copies that compute the homogeneous components up to degree i. ∎

Lemma 3.3 (cf. Lemma 2.4 in [3]). *Suppose $P(x,y) \in \mathbb{F}[X_n, y]$ can be computed by a skew circuit over $\mathbb{F}$ of size s. Then for any i, $\frac{\partial^i P}{\partial^i y}$ can be computed by a skew circuit of size $O(r \cdot s)$, where $r = \deg_y(P)$.*

Proof. Let $C(x,y)$ be a skew circuit for P of size s. We can compute $C_0(x), C_1(x), \ldots, C_r(x)$ with an $r+1$-output skew circuit of size $O(r \cdot s)$ by evaluating $C(x, a_i)$ at $r+1$ distinct elements $a_1, a_2, \ldots, a_{r+1} \in \mathbb{F}$, and then use linear interpolation. Next we can compute $\frac{\partial^i P}{\partial^i y}$ by adding $O(r^2)$ many gates. Since $r \leq s$, the lemma follows. ∎

Lemma 3.4 (Lemma 2.1 in [13]). *Let $f \in \mathbb{F}[X_n]$ be a non-zero polynomial such that the degree of f in x_i is bounded by r_i, and let $S_i \subseteq \mathbb{F}$ be of size at least $r_i + 1$, for all $i \in [n]$. Then there exists $(s_1, s_2, \ldots, s_n) \in S_1 \times S_2 \times \ldots \times S_n$ with $f(s_1, s_2, \ldots, s_n) \neq 0$.*

Lemma 3.5 (Nisan-Wigderson Design [14]). *Let n, m be integers with $n < 2^m$. There exists a family of sets $S_1, S_2, \ldots, S_n \subseteq [\ell]$, such that (1) $\ell = O(m^2/\log n)$, (2) For each i, $|S_i| = m$, and (3) For every $i \neq j$, $|S_i \cap S_j| \leq \log n$. Furthermore, the above family of sets can be computed deterministically in time poly$(n, 2^\ell)$.*

Berkowitz [15] observes that Samuelson's algorithm [16] for computing the characteristic polynomial, does not use divisions and can be implement in NC^2 (Also see [4]). From this one derives the following statement, sufficient for our purpose:

Proposition 3.6. *The determinant of an $n \times n$ matrix M with integer entries of at most m bits each can be computed in time poly(n, m).*

4. Root Extraction within the Skew Circuit Model

We start with the observation that Theorem 3.1 in [3] can be modified into the following lemma. A proof will appear in the full version of this paper.

Lemma 4.1. *Let $n, s,$ and m be integers with $s \geq n$. Let $P(x,y) \in \mathbb{F}[X_n, y]$ be a non-zero polynomial with $s = L_{skew}(P)$. Let $f \in \mathbb{F}[X_n]$ be a polynomial with $\deg(f) = m$ such that $P(x, f(x)) \equiv 0$. Then $L_{skew}(f) = O(s \cdot rm^{r+1})$, where $r = \deg_y(P)$.*

Comparing this with the $s \cdot 2^{O(\log^2 m)} r^{4+\log m}$ bound of Lemma 2.10, which can be bounded by $s \cdot m^{O(\log m + \log r)}$, we see that we get a significant improvement for any $m << 2^r$.

Let us briefly indicate the idea behind the proof of Lemma 2.10. Similar as was done in [3], we want to approximate f up to some degree k, i.e. find a polynomial g with $H_{\leq k}(f) = H_{\leq k}(g)$. In [3] this is done in increments of k by one. This will not be good enough for our purpose. Due to the nature of the skew circuit model, typically any increment of k requires duplication of previously constructed circuitry, leading to an overall exponential blowup by a factor of 2^m. The solution is to aim for a faster *convergence rate* that doubles k in stages. This way, one can keep circuit blow-up due to duplications more or less in check.

We now proceed with the proof of Lemma 2.10. In the following, for any polynomial q the homogeneous component $H_t[q]$ will also be denoted by q_t.

Lemma 4.2. *Let $P \in \mathbb{F}[X_n, y]$ be such that $\deg_y(P) = r$. Write $P = \sum_{i=0}^{r} C_i(x) y^i$, and let $P'(x,y) = \sum_{i=0}^{r} i C_i(x) y^{i-1}$. Let $f \in \mathbb{F}[X_n]$ be such that $P(x, f(x)) = 0$ and $P'(0, f(0)) = \xi_0 \neq 0$. Let $k \geq 1$ be an integer. Suppose $g \in \mathbb{F}[X_n]$ satisfies $H_{\leq k}[g] = H_{\leq k}[f]$. Then for any $1 \leq j \leq k$,*

$$f_{k+j} = g_{k+j} - \frac{1}{\xi_0} \left(P(x,g)_{k+j} + \sum_{i=1}^{j-1} (f_{k+i} - g_{k+i}) P'(x,g)_{j-i} \right).$$

Proof. Let $h = (f_{k+1} - g_{k+1}) + \ldots + (f_{2k} - g_{2k})$. Then

$$\begin{aligned} 0 &= H_{\leq 2k}[P(x, f(x))] \\ &= H_{\leq 2k}[P(x, g+h)] \\ &= H_{\leq 2k}[\sum_{i=0}^{r} C_i(x) \, (g+h)^i] \\ &= H_{\leq 2k}[\sum_{i=0}^{r} C_i(x) \left(g^i + i \cdot g^{i-1} \cdot h\right)] \\ &= H_{\leq 2k}[P(x,g) + P'(x,g) \cdot h] \end{aligned}$$

Let $1 \leq j \leq k$ be given. By the above

$$\begin{aligned} 0 &= P(x,g)_{k+j} + \sum_{i=1}^{j} (f_{k+i} - g_{k+i}) P'(x,g)_{j-i} \\ &= P(x,g)_{k+j} + (f_{k+j} - g_{k+j}) P'(x,g)_0 + \sum_{i=1}^{j-1} (f_{k+i} - g_{k+i}) P'(x,g)_{j-i} \end{aligned}$$

Since $P'(x,g)_0 = P'(0, g(0)) = P'(0, f(0))$, the lemma follows. ∎

Applying the above lemma for $g = H_{\leq k}(f)$ yields the following corollary:

Corollary 4.3. *Let $P \in \mathbb{F}[X_n, y]$ be such that $\deg_y(P) = r$. Write $P = \sum_{i=0}^r C_i(x)y^i$, and let $P'(x,y) = \sum_{i=0}^r iC_i(x)y^{i-1}$. Let $f \in \mathbb{F}[X_n]$ be such that $P(x, f(x)) = 0$ and $P'(0, f(0)) = \xi_0 \neq 0$. Let $k \geq 1$ be an integer. Then for any $1 \leq j \leq k$,*

$$f_{k+j} = -\frac{1}{\xi_0}\left(P(x,g)_{k+j} + \sum_{i=1}^{j-1} f_{k+i} \cdot P'(x,g)_{j-i}\right), \tag{4.1}$$

where $g = H_{\leq k}[f]$.

Lemma 4.4. *Let $P \in \mathbb{F}[X_n, y]$ be such that $\deg_y(P) = r$. Write $P = \sum_{i=0}^r C_i(x)y^i$, and let $P'(x,y) = \sum_{i=0}^r iC_i(x)y^{i-1}$. Let $f \in \mathbb{F}[X_n]$ be such that $P(x, f(x)) = 0$ and $P'(0, f(0)) = \xi_0 \neq 0$. Let $k \geq 1$ be an integer. Let*

$$\mathcal{P} = \{P(x,g)_j : 1 \leq j \leq 2k\} \cup \{P'(x,g)_j : 1 \leq j \leq k-1\},$$

where $g = H_{\leq k}[f]$. Suppose any polynomial in $\mathcal{P}$ can be computed by a single output ABP of size at most B. Then for any $1 \leq j \leq k$, there exist a $(j+1)$-output ABP Φ_j computing $1, f_{k+1}, f_{k+2}, \ldots, f_{k+j}$ of size at most $2Bj^2$.

Proof. We prove the lemma by induction on j. For $j = 1$, we see by Corollary 4.3 that $f_{k+1} = -\frac{1}{\xi_0}P(x,g)_{k+j}$. Hence we have an single output ABP computing f_{k+1} of size at most B. This means we certainly can compute 1 and f_{k+1} by means of a 2-output ABP of size at most $2B$.

Now suppose $1 < j < k$. By induction hypothesis we have a j-output ABP Φ_{j-1} of size at most $2B(j-1)^2$ computing $1, f_{k+1}, f_{k+2}, \ldots, f_{k+j-1}$. The ABP Φ_j is constructed from Φ_{j-1} by first of all passing along all of $1, f_{k+1}, f_{k+2}, \ldots, f_{k+j-1}$ to the outputs. Then by drawing wires from each of these we can compute f_{k+j} according to Equation (4.1). For this we use a new copy of a single output ABP computing some polynomial in $\mathcal{P}$ exactly j times. This construction can be implemented such that $size(\Phi_j) \leq size(\Phi_{j-1}) + jB + j + 1 \leq 2Bj^2$ (For this exact count we use that the cross wires are not actually needed, since we can identify nodes). ■

Lemma 4.5. *Let n, s, r, m and be integers with $s \geq n$. Let $P \in \mathbb{F}[X_n, y]$ be a non-zero polynomial with $\deg_y(P) = r$. Write $P = \sum_{i=0}^r C_i(x)y^i$, and let $P'(x,y) = \sum_{i=0}^r iC_i(x)y^{i-1}$. Assume that both P and P' can be computed by skew circuits of size at most s over $\mathbb{F}$. Let $f \in \mathbb{F}[X_n]$ be a polynomial with $\deg(f) = m$ such that $P(x, f(x)) \equiv 0$ and $P'(0, f(0)) \neq 0$. Then f can be computed by a skew circuit of size at most $s \cdot 2^{O(\log^2 m)} r^{3+\log m}$.*

Proof. We compute f in at most $\lceil \log m \rceil$ stages. At stage i we construct an ABP Ψ_i computing $H_{\leq 2^i}[f]$ of size s_i.

For stage $i = 0$, since $H_{\leq 2^i}[f]$ is an affine linear form in n variables, Ψ_0 can be constructed with $s_0 = O(n)$.

We now describe stage i, for $i > 0$. Let $g = H_{\leq 2^{i-1}}[f]$. In the previous stage an ABP Ψ_{i-1} was constructed for g of size s_{i-1}.

We claim $P(x,g)$ and $P'(x,g)$ can be computed by an ABP of size $O(rs_{i-1} + r^2 s)$. Namely, like in proof of Lemma 3.3, we have for any i, an ABP of size $O(rs)$ computing $C_i(x)$. Using r copies of the ABP computing g we can then compute $\sum_{i=0}^r C_i(x)g^i$ with size $O(rs_{i-1} + r^2 s)$. Similarly, for $P'(x,g)$.

Hence, by Proposition 3.2 and Proposition 3.1, for any $j \leq 2^i$, $P(x,g)_j$ can be computed by an ABP of size $O(2^i(rs_{i-1}+r^2s))$. Similarly, for any $j \leq 2^{i-1}$, $P'(x,g)_j$ can be computed by an ABP of size $O(2^{i-1}(rs_{i-1}+r^2s))$.

Therefore, we can apply Lemma 4.4 with $k = 2^{i-1}$ and $B := O(2^i(rs_{i-1}+r^2s))$. This gives us an ABP Φ_{2k} computing $f_{k+1}, f_{k+2}, \ldots, f_{2k}$ of size at most $2Bk^2$. Combining Ψ_k and Φ_{2k} to add all components of f gives us the ABP Ψ_{2k} computing $H_{\leq 2k}[f]$ of size $O(2^{3i}(rs_{i-1}+r^2s)+s_{i-1})$. We can thus bound $s_i \leq \alpha r 2^{3i} \cdot (s_{i-1}+rs)$, for some absolute constant $\alpha > 1$. From this, one gets that $s_i \leq s \cdot \beta^{i^2+1} r^{i+2}$, for some absolute constant $\beta > 1$.

Taking i $= \lceil \log m \rceil$, we see there exists an ABP computing f with size bounded by $s \cdot 2^{O(\log^2 m)} r^{3+\log m}$. Applying Proposition 3.1 completes the proof. ∎

4.1. Proof of Lemma 2.10

Write $P = \sum_{i=0}^{r} C_i(x) y^i$ with $C_r(x) \not\equiv 0$. Let $P^i(x,y) = \frac{\partial^i P}{\partial^i y}$. Then $P^r(x,y) = r! \cdot C_r(x)$. Since the characteristic of $\mathbb{F}$ is zero, $r! \neq 0$, and hence $P^r(x,f(x)) \not\equiv 0$. By assumption, $P^0(x,f(x)) \equiv 0$. Let i be the smallest number such that $P^i(x,f(x)) \not\equiv 0$. Then $0 < i \leq r$, and $P^{i-1}(x,f(x)) \equiv 0$. We have that there exists $x_0 \in \mathbb{F}$ such that $P^i(x_0,f(x_0)) \neq 0$.

Let $Q(x,y) = P^{i-1}(x+x_0,y)$, and let $g = f(x+x_0)$. Q is computable by a skew circuit of size $O(r \cdot s)$ by Lemma 3.3. Let $Q' = \frac{\partial Q}{\partial y}$. Observe $Q'(x,y) = P^i(x+x_0,y)$. Q is a nonzero polynomial such that $Q(x,g(x)) = P^{i-1}(x+x_0,f(x+x_0)) \equiv 0$, and $Q'(0,g(0)) = P^i(x_0,f(x_0)) \neq 0$. We apply Lemma 4.5 and obtain a skew circuit Ψ computing $g(x)$ of size $s \cdot 2^{O(\log^2 m)} r^{4+\log m}$. From this a skew circuit computing f is obtained that is at most a constant factor larger by performing the substitution $x := x - x_0$ within Ψ. ∎

5. Constructing a Generator from a Hard Polynomial

With the "Root Extraction" Lemmas 2.10 and 4.1 proved, the following lemma follows by the technique of Lemma 7.6 in [9], which was also employed to prove Lemma 4.1 in [3]. We use the notation that for a set $S \subseteq [\ell]$ of size m, and a vector of variables $y = (y_1, y_2, \ldots, y_l)$, $f(y|_S)$ denotes $f(y_{s_1}, y_{s_2}, \ldots, y_{s_m})$, where $s_1, s_2, \ldots, s_m$ is an arbitrary ordering of the elements of S. A proof of the lemma will appear in the full version of this paper.

Lemma 5.1. *Let n, r and s be integers, and let $g \in \mathbb{F}[X_n]$ be a non-zero polynomial with individual degrees bounded by r with $L_{skew}(g) = s \geq n$. Let $m > \log n$ be an integer and let $S_1, S_2, \ldots, S_n \subseteq [\ell]$ be given by Lemma 3.5, so that $\ell = O(m^2/\log n)$, $|S_i| = m$, and $|S_i \cap S_j| \leq \log n$. Let $f \in \mathbb{F}[z_1, z_2, \ldots, z_m]$ be a multilinear polynomial such that $g(f(y|_{S_1}), f(y|_{S_2}), \ldots, f(y|_{S_n})) \equiv 0$, where $y = (y_1, y_2, \ldots, y_\ell)$ is a vector of variables. Then $L_{skew}(f) \leq sn \cdot \min(2^{c_1(\log^2 m)} r^{4+\log m}, c_2 \cdot rm^{r+1})$, for absolute constants $c_1, c_2 > 1$.*

5.1. Proof of Theorem 2.6 and 2.7

Proof. We first consider Theorem 2.7. Suppose $\{f_m\}$ is an explicit multilinear family with $\mathrm{dc}(f_m) = m^{\omega(\log m)}$. Consider some large enough n. Set $m = \lceil 2^{\frac{1}{2}\sqrt{\log n}} \rceil$. Construct the Nisan-Wigderson design $S_1, S_2, \ldots, S_n$ as in Lemma 5.1 with $\ell(n) = O(m^2/\log n) = O(n^{1/\sqrt{\log n}})$. We claim the required $(\ell(n), n)$-generator G_n can be given by

$$G_n(y_1, y_2, \ldots, y_{\ell(n)}) \triangleq (f_m(y|_{S_1}), f_m(y|_{S_2}), \ldots, f_m(y|_{S_n})),$$

To verify this, consider any $k(n) \in n^{O(1)}$ and $r(n) \in 2^{O(\sqrt{\log n})}$, and arbitrary $k(n) \times k(n)$ matrix $M(x)$ with entries in $\mathcal{A}_{\mathbb{F}}(X_n)$. Let $g = \det(M(x))$. Assume the individual degrees of g are bounded by $r(n) = poly(m)$. Observe it suffices to verify that if $g \not\equiv 0$, then $\det(M(G_n(y))) \not\equiv 0$. Due to [4], we know g has a skew circuit over $\mathbb{F}$ of size at most $O(n \cdot k(n)^6) \leq n^d$, for some constant d (provided n is large enough). Hence by Lemma 5.1, if $\det(M(G_n(y))) \equiv 0$, we obtain a skew circuit over $\mathbb{F}$ for f_m of size at most $n^{d+1} \cdot 2^{c_1(\log^2 m)} r(n)^{4+\log m} \leq 2^{4(d+1)\log^2 m} \cdot 2^{c_1(\log^2 m)} r(n)^{4+\log m}$. Since $r(n) = poly(m)$ and n is assumed to be large enough, this contradicts the hardness of f_m. (Here we use $\mathrm{dc}(f_m) = O(L_{skew}(f_m)))$.

For Theorem 2.6 one argues similarly, but with $m := \lceil n^\epsilon \rceil$. We bound the size of the skew circuit for f_m by $c_2 n^{d+1} \cdot r(n) m^{r(n)+1} \leq c_2 r(n) m^{(d+1)/\epsilon + r(n)+1}$. This contradicts the hardness of f_m, assuming $\mathrm{dc}(f_m) = m^{\omega(1)}$, for any constant $0 < \epsilon < 1$ and $r(n) = O(1)$, provided n is large enough.

We now check that in any of the above cases, $\{G_n\}_{n \geq 1}$ is efficiently computable. Given (i, n, e), where $e \in \{0,1\}^{\ell(n)}$, one first constructs the sets $S_1, S_2, \ldots, S_n$. This can be done deterministically in time $2^{O(\ell(n))}$ by Lemma 3.5. Then if for some $j \notin S_i$, $e_j = 1$, return zero. Otherwise, let $c = e|_{S_i}$. Return the coefficient of the monomial $x_1^{c_1} x_2^{c_2} \ldots x_m^{c_m}$ of f_m. Since f_m is explicit, this coefficient can be computed deterministically in time $2^{O(m)}$. Hence the total deterministic time is bounded by $2^{O(\ell(n))}$. ■

Remark 5.2. From the above we see an $(\lceil n^\epsilon \rceil, n)$-generator for $\mathrm{NSMC}^{poly(n)}_{r(n)}(\mathbb{F})$ can be obtained by assuming $\mathrm{dc}(f_m) = m^{\omega(r(m^{1/\epsilon}))}$. For example, assuming $\mathrm{dc}(f_m) = m^{\omega(\log\log m)}$ yields an $(\lceil n^\epsilon \rceil, n)$-generator for $\mathrm{NSMC}^{poly(n)}_{\log\log n}(\mathbb{F})$, for any $0 < \epsilon < 1$.

6. Using the Generator to decide $\mathrm{NSMC}(\mathbb{Q})$ Deterministically

Theorem 6.1. *Let $\ell(n)$ and $r(n)$ be functions of type $\mathbb{N} \to \mathbb{N}$ such that $\log n < \ell(n) < n$, for all large enough n. If there exists an efficiently computable multilinear $(\ell(n), n)$-generator $\{G_n\}_{n \geq 1}$, such that for any $p(n) \in n^{O(1)}$, G_n provides a test for $\mathrm{NSMC}^{p(n)}_{r(n)}(\mathbb{Q})$, for all large enough n, then for any $k(n) \in n^{O(1)}$, $\mathrm{NSMC}^{k(n)}_{r(n)}(\mathbb{Q})$ can be decided deterministically in time $2^{O(\ell(n)\log n + \ell(n)\log r(n))}$, provided coefficients of the input matrix have bit size $n^{O(1)}$.*

Proof. Say G_n is defined over variables $z_1, z_2, \ldots, z_{\ell(n)}$. Consider an arbitrary matrix M of size $k(n)$, with entries in $\mathcal{A}_{\mathbb{Q}}(X_n)$, where coefficients have bit size $n^{O(1)}$, and with individual degrees of $\det(M(x))$ bounded by $r(n)$. We assume wlog. that entries of M are in $\mathcal{A}_{\mathbb{Z}}(X_n)$, since we can multiply out all denominators and still leave bit sizes bounded by $n^{O(1)}$.

For large enough n, by Definition 2.5, $(\exists a \in \mathbb{Q}^n), \det M(a) \neq 0$ iff $(\exists b \in \mathbb{Q}^{\ell(n)}), \det M(G_n(b)) \neq 0$. Let $m = \ell(n)$. We have that $(\exists b \in \mathbb{Q}^m), \det M(G_n(b)) \neq 0$ if and only if $h := \det M(G_n(z)) \not\equiv 0$. Individual degrees of h are at most $nr(n)$. By Lemma 3.4, if $h \not\equiv 0$, then for some $b \in V^m$, $h(b) \neq 0$, where $V = \{0, 1, \ldots, nr(n)\}$. Hence we can use the following test, for any n larger than some fixed threshold depending on k:

Algorithm. Test (input : an instance $M(x)$ of $\mathrm{NSMC}_{r(n)}^{k(n)}(\mathbb{Z})$)

```
(1) Let $V = \{0, 1, \ldots, nr(n)\}$.
(2) For all $b \in V^{\ell(n)}$, compute $v_b := \det(M(G_n(b)))$.
(3) If for all $b \in V^{\ell(n)}$, $v_b = 0$, then Reject else Accept.
```

If the above algorithm accepts, one also knows a non-singular completion. Let us estimate the running time. Since G_n is efficiently computable, for any $b \in V^{\ell(n)}$, $G_n(b)_j$ can be computed in time $2^{O(m)}$. Each entry of $N := M(G_n(b))$ is an integer computable in time $2^{O(m)}$. By Proposition 3.6, $\det(N)$ is computable in time $poly(k(n), 2^{O(m)}) = 2^{O(m)}$. Hence the total time is bounded by $2^{O(m)} \cdot (nr(n)+1)^m = 2^{O(m \log n + m \log r(n))}$. ∎

Using Theorem 6.1, the proofs of Theorem 2.8 and Theorem 2.9 immediately follow from Theorem 2.6 and Theorem 2.7, respectively.

7. Constructing a Hard Polynomial from a Generator

Let $\delta > 0$. We say a function $\ell : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ is *δ-nice* if 1) ℓ is monotone increasing, 2) $\ell(t)^{1+\delta} < t$ and $|\ell(t+1)^{1+\delta} - \ell(t)^{1+\delta}| \leq 1$, for all large enough t, and 3) for all large enough N, given N in unary, we can[2] compute an n such that $N = \lceil \ell(n)^{1+\delta} \rceil$ deterministically in time $2^{O(N)}$.

Theorem 7.1. *Let $\delta > 0$, and let $\ell : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ be a δ-nice function. Given any efficiently computable multilinear $(\lceil \ell(n) \rceil, n)$-generator $\{G_n\}_{n \geq 1}$, we can construct an explicit multilinear family $\{g_N\}_{N \geq 1}$, such that if for some integer $d > 0$, G_n provides a test for $\mathrm{NSMC}_1^{n^d}(\mathbb{F})$ for all large enough n, then for all large enough N, $dc(g_N) > \ell^{-1}(N^{1/(1+\delta)})^d$, over the field $\mathbb{F}$.*

Proof. Consider some large enough N. Let n be such that $N = \lceil \ell(n)^{1+\delta} \rceil$ (such an n can be found in time $2^{O(N)}$). Let $m = \lceil \ell(n) \rceil$. We have that $N \leq n$. Let $V = \{1, 2, \ldots, N+1\} \subseteq \mathbb{F}$. Similarly[3] as in [17], define the polynomial $g_N(x_1, x_2, \ldots, x_N) = \sum_{I \subseteq [1,N]} c_I \prod_{i \in I} x_i$, where c_I is taken to be an *integer* nonzero solution of the following system of linear equations:

$$\sum_{I \subseteq [1,N]} c_I \prod_{i \in I} G_n(a_1, a_2, \ldots, a_m)_i = 0, \tag{7.1}$$

for all $a \in V^m$. These are $(N+1)^m$ equations in 2^N variables. Provided n is large enough, $m \log(N+1) < N$, and hence there exists a nonzero solution over $\mathbb{F}$. The technical conditions placed on $\ell(t)$ ensure g_N is defined for all large enough N. Below we will argue how to compute an integer solution within time $2^{O(N)}$, so that g_N is explicit in the sense of Definition 2.2.

[2] Note: conditions 1) and 2) imply the n in condition 3) always exists, provided N is large enough.

[3] Agrawal [17] works with a different notion of a generator, and does not demand integer coefficients for explicitness.

For the purpose of contradiction, suppose that $\mathrm{dc}(g_N) \leq n^d$. Hence we can write $g_N = \det(M)$, where M is an $n^d \times n^d$ matrix with entries in $\mathcal{A}_{\mathbb{F}}(X_N)$. The entries of M are elements of $\mathcal{A}_{\mathbb{F}}(X_n)$, since $\mathcal{A}_{\mathbb{F}}(X_N) \subseteq \mathcal{A}_{\mathbb{F}}(X_n)$. Since $\mathbb{F}$ is an infinite field and $g_N \not\equiv 0$, there exists $a \in \mathbb{F}^n$ such that $\det(M(a)) = g_N(a_1, a_2, \ldots, a_N) \neq 0$. The individual degrees of g_N are bounded by one. Hence, by Definition 2.5, there exists $b \in \mathbb{F}^m$ such that $g_N(G_n(b)_1, G_n(b)_2, \ldots, G_n(b)_N) = \det(M(G_n(b))) \neq 0$. This implies $h \not\equiv 0$, where $h(z) := g_N(G_n(z)_1, G_n(z)_2, \ldots, G_n(z)_N)$. Observe that individual degrees of h are bounded by N. Hence by Lemma 3.4, there exists $b' \in V^m$ such that $h(b') \neq 0$, but this contradicts (7.1). Therefore $\mathrm{dc}(g_N) > n^d \geq \ell^{-1}(N^{1/(1+\delta)})^d$, for all large enough N.

We now argue how to obtain an integer solution to (7.1). Since G_n is efficiently computable, we can compute any coefficient $G_n(a_1, a_2, \ldots, a_m)_i$ by summing over all 2^m monomials. This takes time $2^{O(m)}$. We write (7.1) as $Ax = 0$, for an $r \times 2^N$ matrix A, with integer coefficients of bit size $2^{O(m)}$ and $r = (N+1)^m$. To construct A takes time $2^{O(N)}$.

First, we want to find an independent set S of $rank(A)$ many rows of A, and then extend S to an independent set of size 2^N. Let $e_1, e_2, \ldots, e_{2^N}$ denote the standard basis row-vectors of $\mathbb{F}^{2^N}$. One can do this as follows:

```
(1) let $v_i$ equal row $i$ of $A$, for $i \in [r]$, and let $v_{r+i} = e_i$, for $i \in [2^N]$.
(2) let $S = \emptyset$
(3) for $i = 1$ to $r + 2^N$
(4)     let $S' = S \cup \{v_i\}$
(5)     compute $\beta = \det(BB^T)$, where $B$ is the $|S'| \times 2^N$ matrix of rows in $S'$.
(6)     if $\beta \neq 0$, then set $S = S'$
```

By the Binet-Cauchy Theorem, $\det(BB^T) = \sum_{I \subseteq 2^N, |I| = |S'|} [\det(B_I)]^2$, where B_I is the $|S'| \times |S'|$ matrix consisting of the columns in I of B. Hence $\beta \neq 0$ if and only if there exists a set I of $|S'|$ independent columns in B. The latter holds if and only if S' is an independent set. The above procedure therefore maintains the invariant that after execution of line 6, S is an independent set with $\{v_1, \ldots, v_i\} \subseteq span(S)$ (We use the convention that $\emptyset$ is an independent set with $span(\emptyset) = \{0\}$). This implies that after the rth iteration, S contains $rank(A)$ many rows of A, and after the final iteration, S is a basis.

Entries of BB^T have bit size $2^{O(N)}$. By Proposition 3.6, $\det(BB^T)$ can be computed in time $2^{O(N)}$. Hence the above procedure takes time $2^{O(N)}$ in total.

Let B be the matrix consisting of the rows in S computed by the above procedure. B is computable in time $2^{O(N)}$. Consider the adjugate $adj(B)$. It satisfies $B \cdot adj(B) = \det(B)I$. Hence we can pick a nonzero column from $adj(B)$ that is a solution to the original system (7.1). The entry $adj(B)_{ij} = (-1)^{i+j} M_{ji}$, where M_{ij} is the determinant of the matrix B with rows i and j removed. The latter is an integer, and by Proposition 3.6 it is computable in time $2^{O(N)}$. ∎

One proves Theorem 2.11 using Theorem 7.1 with $\ell(t) = t^{\epsilon}$, and selecting a small $\delta > 0$ such that $\epsilon(1+\delta) \in \mathbb{Q} \cap (0,1)$. Then ℓ is δ-nice. This yields an explicit multilinear family $\{g_N\}_{N \geq 1}$, such that for any d, for all large enough N, $\mathrm{dc}(g_N) > N^{d/(\epsilon(1+\delta))}$. Hence $\mathrm{dc}(g_N) = N^{\omega(1)}$.

For Theorem 2.12, assume wlog. $\{G_n\}_{n \geq 1}$ is an efficiently computable multilinear $(\lceil \ell(n) := c \cdot n^{1/\sqrt{\log n}} \rceil, n)$-generator, for a constant $c \in \mathbb{Z}_{>0}$. Then $\ell^{-1}(n) = 2^{\log^2(n/c)}$, and

ℓ is δ-nice, for $\delta = 1$. Theorem 7.1 yields an explicit multilinear family $\{g_N\}_{N\geq 1}$, such that for any d, for all large enough N, $\mathrm{dc}(g_N) > 2^{d\cdot\log^2(\frac{N^{1/2}}{c})}$. Hence $\mathrm{dc}(g_N) = N^{\omega(\log N)}$.

References

[1] L. Lovász. On determinants, matching, and random algorithms. In *FCT'79: Fundamentals of Computation Theory*, pages 565–574, 1979.

[2] J. Edmonds. Systems of distinct representatives and linear algebra. *Journal of Research of the National Bureau of Standards*, 71B:241–245, 1967.

[3] Z. Dvir, A. Shpilka, and A Yehudayoff. Hardness-randomness tradeoffs for bounded depth circuits. In *Proceedings of the 40th Annual STOC*, pages 741–748, 2008.

[4] M. Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science*, 1997(Article 5), 1997.

[5] L. Valiant. Completeness classes in algebra. Technical Report CSR-40-79, Dept. of Computer Science, University of Edinburgh, April 1979.

[6] J.T. Schwartz. Fast probabilistic algorithms for polynomial identities. *J. Assn. Comp. Mach.*, 27:701–717, 1980.

[7] R. Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Manipulation (EUROSAM '79)*, volume 72 of *Lect. Notes in Comp. Sci.*, pages 216–226, Marseilles, June 1979. Springer Verlag.

[8] N.Saxena G. Ivanyos, M. Karpinksi. Deterministic polynomial time algorithms for matrix completion problems. Technical Report ECCC TR09-58, Electronic Colloquium in Computational Complexity, 2009.

[9] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity testing means proving circuit lower bounds. *Computational Complexity*, 13(1–2):1–44, 2004.

[10] T. Mignon and N. Ressayre. A quadratic bound for the determinant and permanent problem. *International Mathematics Research Notices*, pages 4241–4253, 2004.

[11] P. Bürgisser, M. Claussen, and M.A. Shokrollahi. *Algebraic Complexity Theory.* Springer Verlag, 1997.

[12] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12:641–644, 1983.

[13] N. Alon. Combinatorial nullstellensatz. *Combinatorics, Probability and Computing*, 8(1–2):7–29, 1999.

[14] N. Nisan and A. Wigderson. Hardness versus randomness. *J. Comp. Sys. Sci.*, 49:149–167, 1994.

[15] S. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inf. Proc. Lett.*, 18:147–150, 1984.

[16] P.A. Samuelson. A method of determining explicitly the coefficients of the characteristic polynomial. *Annals of Mathematical Statistics*, 13:424–429, 1942.

[17] M. Agrawal. Proving lower bounds via pseudo-random generators. In *Proc. 25th Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 92–105, 2005.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 477-488
www.stacs-conf.org

ON EQUATIONS OVER SETS OF INTEGERS

ARTUR JEŻ [1] AND ALEXANDER OKHOTIN [2,3]

[1] Institute of Computer Science, University of Wrocław
E-mail address: aje@ii.uni.wroc.pl

[2] Academy of Finland

[3] Department of Mathematics, University of Turku, Finland
E-mail address: alexander.okhotin@utu.fi

Abstract. Systems of equations with sets of integers as unknowns are considered. It is shown that the class of sets representable by unique solutions of equations using the operations of union and addition $S+T=\{m+n \mid m \in S,\, n \in T\}$ and with ultimately periodic constants is exactly the class of hyper-arithmetical sets. Equations using addition only can represent every hyper-arithmetical set under a simple encoding. All hyper-arithmetical sets can also be represented by equations over sets of natural numbers equipped with union, addition and subtraction $S \dot{-} T = \{m-n \mid m \in S,\, n \in T,\, m \geqslant n\}$. Testing whether a given system has a solution is Σ^1_1-complete for each model. These results, in particular, settle the expressive power of the most general types of language equations, as well as equations over subsets of free groups.

1. Introduction

Language equations are equations with formal languages as unknowns. The simplest such equations are the context-free grammars [4], as well as their generalization, the conjunctive grammars [15]. Many other types of language equations have been studied in the recent years, see a survey by Kunc [11], and most of them were found to have strong connections to computability. In particular, for equations with concatenation and Boolean operations it was shown by Okhotin [19, 17] that the class of languages representable by their unique (least, greatest) solutions is exactly the class of recursive (r.e., co-r.e.) sets. A computationally universal equation of the simplest form was constructed by Kunc [10], who proved that the greatest solution of the equation $XL = LX$, where $L \subseteq \{a,b\}^*$ is a finite constant language, may be co-r.e.-complete.

A seemingly trivial case of language equations over a *unary alphabet* $\Omega = \{a\}$ has recently been studied. Strings over such an alphabet may be regarded as natural numbers,

1998 ACM Subject Classification: F.4.3 (Formal languages), F.4.1 (Mathematical logic).

Key words and phrases: Language equations, computability, arithmetical hierarchy, hyper-arithmetical hierarchy.

Research supported by the Polish Ministry of Science and Higher Education under grants N N206 259035 2008–2010 and N N206 492638 2010–2012, and by the Academy of Finland under grant 134860.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE
27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2478

and languages accordingly become sets of numbers. As established by the authors [8], these equations are as powerful as language equations over a general alphabet: a set of natural numbers is representable by a unique solution of a system with union and elementwise addition if and only if it is recursive. Furthermore, even without the union operation these equations remain almost as powerful [9]: for every recursive set $S \subseteq \mathbb{N}$, its encoding $\sigma(S) \subseteq \mathbb{N}$ satisfying $S = \{n \mid 16n + 13 \in \sigma(S)\}$ can be represented by a unique solution of a system using addition only, as well as ultimately periodic constants. At the same time, as shown by Lehtinen and Okhotin [12], some recursive sets are not representable without an encoding.

Equations over sets of numbers are, on one hand, interesting on their own as a basic mathematical object. On the other hand, these equations form a very special case of language equations with concatenation and Boolean operations, which turned out to be as hard as the general case, and this is essential for understanding language equations. However, it must be noted that these cases do not exhaust all possible language equations. The recursive upper bound on unique solutions [19] is applicable only to equations with *continuous* operations on languages, and using the simplest non-continuous operations, such as homomorphisms or quotient [18], leads out of the class of recursive languages. In particular, a quotient with regular constants was used to represent all sets in the arithmetical hierarchy [18].

The task is to find a natural limit of the expressive power of language equations, which would not assume continuity of operations. As long as operations on languages are expressible in first-order arithmetic (which is true for every common operation), it is not hard to see that unique solutions of equations with these operations always belong to the family of *hyper-arithmetical sets* [14, 20, 21]. This paper shows that this obvious upper bound is in fact reached already in the case of a unary alphabet.

To demonstrate this, two abstract models dealing with sets of numbers shall be introduced. The first model are equations over sets of natural numbers with addition $S + T = \{m + n \mid m \in S,\ n \in T\}$ and subtraction $S \dot{-} T = \{m - n \mid m \in S,\ n \in T,\ m \geqslant n\}$ (corresponding to concatenation and quotient of unary languages), as well as set-theoretic union. The other model has sets of integers, including negative numbers, as unknowns, and the allowed operations are addition and union. The main result of this paper is that unique solutions of systems of either kind can represent every *hyper-arithmetical* set of numbers.

The base of the construction is the authors' earlier result [8] on representing every recursive set by equations over sets of natural numbers with union and addition. In Section 2, this result is adapted to the new models introduced in this paper. The next task is representing every set in the arithmetical hierarchy, which is achieved in Section 3 by simulating existential and universal quantifiers over a recursive set. These arithmetical sets are then used in Section 4 as constants for the construction of equations representing hyper-arithmetical sets. Finally, the constructed equations are encoded in Section 5 using equations over sets of integers with addition only and periodic constant sets.

This result brings to mind a study by Robinson [20], who considered equations, in which the unknowns are functions from $\mathbb{N}$ to $\mathbb{N}$, the only constant is the successor function and the only operation is superposition, and proved that a function is representable by a unique solution of such an equation if and only if it is hyper-arithmetical. Though these equations deal with objects different from sets of numbers, there is one essential thing in common: in both results, unique solutions of equations over second-order arithmetical objects represent hyper-arithmetical sets.

Some more related work can be mentioned. Halpern [5] studied the decision problem of whether a formula of Presburger arithmetic with set variables is true for all values of these set variables, and showed that it is Π^1_1-complete. The equations studied in this paper can be regarded as a small fragment of Presburger arithmetic with set variables.

Another relevant model are languages over free groups, which have been investigated, in particular, by Anisimov [3] and by d'Alessandro and Sakarovitch [2]. Equations over sets of integers are essentially equations for languages over a monogenic free group.

An important special case of equations over sets of numbers are *expressions* and *circuits* over sets of numbers, which are equations without iterated dependencies. Expressions and circuits over sets of natural numbers were studied by McKenzie and Wagner [13], and a variant of these models defined over sets of integers was investigated by Travers [22].

2. Equations and their basic expressive power

The subject of this paper are systems of equations of the form

$$\begin{cases} \varphi_1(X_1, \ldots, X_n) & = & \psi_1(X_1, \ldots, X_n) \\ & \vdots & \\ \varphi_m(X_1, \ldots, X_n) & = & \psi_m(X_1, \ldots, X_n) \end{cases}$$

where $X_i \subseteq \mathbb{Z}$ are unknown sets of integers, and the expressions φ_i and ψ_i use such operations as union, intersection, complementation, as well as the main arithmetical operation of elementwise addition of sets, defined as $S + T = \{m + n \mid m \in S, \ n \in T\}$. Subtraction $S - T = \{m - n \mid m \in S, \ n \in T\}$ shall be occasionally used. The constant sets contained in a system sometimes will be singletons only, sometimes any ultimately periodic constants will be allowed (a set of integers $S \subseteq \mathbb{Z}$ is *ultimately periodic* if there exist numbers $d \geqslant 0$ and $p \geqslant 1$, such that $n \in S$ if and only if $n + p \in S$ for all n with $|n| \geqslant d$), and in some cases the constants will be drawn from wider classes of sets, such as all recursive sets. Systems over sets of natural numbers shall have subsets of $\mathbb{N}$ both as unknowns and as constant languages; whenever subtraction is used in such equations, it will be used in the form $S \dot{-} T = (S - T) \cap \mathbb{N}$.

Consider systems with a unique solution. Every such system can be regarded as a specification of a set, and for every type of systems there is a natural question of what kind of sets can be represented by unique solutions of these systems. For equations over sets of natural numbers, these are the recursive sets:

Proposition 1 (Jeż, Okhotin [8, Thm. 4])**.** *The family of sets of natural numbers representable by unique solutions of systems of equations of the form $\varphi_i(X_1, \ldots, X_n) = \psi_i(X_1, \ldots, X_n)$ with union, addition and singleton constants, is exactly the family of recursive sets.*

Turning to the more general cases of equations over sets of integers and of equations over sets of natural numbers with subtraction, an upper bound on their expressive power can be obtained by reformulating a given system in the notation of first-order arithmetic.

Lemma 1. *For every system of equations in variables $X_1, \ldots X_n$ using operations expressible in first-order arithmetic there exists an arithmetical formula $Eq(X_1, \ldots, X_n)$, where $X_1, \ldots, X_n$ are free second-order variables, such that $Eq(S_1, \ldots, S_n)$ is true if and only if $X_i = S_i$ is a solution of the system.*

Constructing this formula is only a matter of reformulation. As an example, an equation $X_i = X_j + X_k$ is represented by $(\forall n)\big[n \in X_i \leftrightarrow (\exists n')(\exists n'')n = n' + n'' \wedge n' \in X_j \wedge n'' \in X_k\big]$.

Now consider the following formulae of second-order arithmetic:

$$\varphi(x) = (\exists X_1)\dots(\exists X_n)Eq(X_1,\dots,X_n) \wedge x \in X_1$$
$$\varphi'(x) = (\forall X_1)\dots(\forall X_n)Eq(X_1,\dots,X_n) \to x \in X_1$$

The formula $\varphi(x)$ represents the membership of x in *any* solution of the system, while $\varphi'(x)$ states that *every* solution of the system contains x. Since, by assumption, the system has a unique solution, these two formulae are equivalent and each of them specifies the first component of this solution. Furthermore, φ and φ' belong to the classes Σ^1_1 and Π^1_1, respectively, and accordingly the solution belongs to the class $\Delta^1_1 = \Sigma^1_1 \cap \Pi^1_1$, known as the class of *hyper-arithmetical sets* [14, 21].

Lemma 2. *For every system of equations in variables $X_1, \dots X_n$ using operations and constants expressible in first-order arithmetic that has a unique solution $X_i = S_i$, the sets S_i are hyper-arithmetical.*

Though this looks like a very rough upper bound, this paper actually establishes the converse, that is, that every hyper-arithmetical set is representable by a unique solution of such equations. The result shall apply to equations of two kinds: over sets of integers with union and addition, and over sets of natural numbers with union, addition and subtraction. In order to establish the properties of both families of equations within a single construction, the next lemma introduces a general form of systems that can be converted to either of the target types of systems:

Lemma 3. *Consider any system of equations $\varphi(X_1,\dots,X_m) = \psi(X_1,\dots,X_m)$ and inequalities $\varphi(X_1,\dots,X_m) \subseteq \psi(X_1,\dots,X_m)$ over sets of natural numbers that uses the following operations: union; addition of a recursive constant; subtraction of a recursive constant; intersection with a recursive constant. Assume that the system has a unique solution $X_i = S_i \subseteq \mathbb{N}$. Then there exist:*

1. *a system of equations over sets of natural numbers in variables $X_1,\dots,X_m,Y_1,\dots,Y_{m'}$ using the operations of addition, subtraction and union and singleton constants, which has a unique solution with $X_i = S_i$;*
2. *a system of equations over sets of integers in variables $X_1,\dots,X_m,Y_1,\dots,Y_{m'}$ using the operations of addition and union, singleton constants and the constants $\mathbb{N}$ and $-\mathbb{N}$, which has a unique solution with $X_i = S_i$.*

Inequalities $\varphi \subseteq \psi$ can be simulated by equations $\varphi \cup \psi = \psi$. For equations over sets of natural numbers, each recursive constant is represented according to Proposition 1, and this is sufficient to implement each addition or subtraction of a recursive constant by a large subsystem using only singleton constants. In order to obtain a system over sets of integers, a straightforward adaptation of Proposition 1 is needed:

Lemma 3.1. *For every recursive set $S \subseteq \mathbb{N}$ there exists a system of equations over sets of integers in variables $X_1,\dots,X_n$ using union, addition, singleton constants and constant $\mathbb{N}$, such that the system has a unique solution with $X_1 = S$.*

This is essentially the system given by Proposition 1, with additional equations $X_i \subseteq \mathbb{N}$.

Now a difference $X \dot{-} R$ for a recursive constant $R \subseteq \mathbb{N}$ shall be represented as $(X + (-R)) \cap \mathbb{N}$, where the set $-R = \{-n \mid n \in R\}$ is specified by taking a system for R and applying the following transformation:

Lemma 3.2 (Representing sets of opposite numbers)**.** *Consider a system of equations over sets of integers, in variables $X_1, \ldots, X_n$, using union and addition, and any constant sets, which has a unique solution $X_i = S_i$. Then the same system, with each constant $C \subseteq \mathbb{Z}$ replaced by the set of the opposite numbers $-C$, has the unique solution $X_i = -S_i$.*

The last step in the proof of Lemma 3 is eliminating intersection with recursive constants. This is done as follows:

Lemma 3.3 (Intersection with constants)**.** *Let $R \subseteq \mathbb{N}$ be a recursive set. Then there exists a system of equations over sets of natural numbers using union, addition and singleton constants, which has variables $X, Y, Y', Z_1, \ldots, Z_m$, such that the set of solutions of this system is*

$$\big\{ (X = S,\ Y = S \cap R,\ Y' = S \cap \overline{R},\ Z_i = S_i) \bigm| S \subseteq \mathbb{N} \big\},$$

where $S_1, \ldots, S_m$ are some fixed sets.

In plain words, the constructed system works as if an equation $Y = X \cap R$ (and also as another equation $Y' = X \cap \overline{R}$, which may be ignored). This completes the transformations needed for Lemma 3.

The last basic element of the construction is representing a set of integers (both positive and negative) by first representing its positive and negative subsets individually:

Lemma 4 (Assembling positive and negative subsets)**.** *Let sets $S \cap \mathbb{N}$ and $(-S) \cap \mathbb{N}$ be representable by unique solutions of equations over sets of integers using union, addition, and ultimately periodic constants. Then S is representable by equations over integers using only union, addition and ultimately periodic constants.*

3. Representing the arithmetical hierarchy

Each arithmetical set can be represented by a recursive relation with a quantifier prefix, and arithmetical sets form the *arithmetical hierarchy* based on the number of quantifier alternations in such a formula. The bottom of the hierarchy are the recursive sets, and every next level is comprised of two classes, Σ^0_k or Π^0_k, which correspond to the cases of the first quantifier's being existential or universal. For every $k \geqslant 1$, a set is in Σ^0_k if it can be represented as

$$\{w \mid \exists x_1 \forall x_2 \ldots Q_k x_k\ R(w, x_1, \ldots, x_k)\}$$

for some recursive relation R, where $Q_k = \forall$ if k is even and $Q_k = \exists$ if k is odd. A set is in Π^0_k if it admits a similar representation with the quantifier prefix $\forall x_1 \exists x_2 \ldots Q_k x_k$. It is easy to see that $\Pi^0_k = \{L \mid \overline{L} \in \Sigma^0_k\}$. The sets Σ^0_1 and Π^0_1 are the recursively enumerable sets and their complements, respectively. The arithmetical hierarchy is known to be strict: $\Sigma^0_k \subset \Sigma^0_{k+1}$ and $\Pi^0_k \subset \Pi^0_{k+1}$ for every $k \geqslant 0$. Furthermore, for every $k \geqslant 1$ the inclusion $\Sigma^0_k \cup \Pi^0_k \subset \Sigma^0_{k+1} \cap \Pi^0_{k+1}$ is proper, i.e., there is a gap between the k-th and $(k+1)$-th level.

For this paper, the definition of arithmetical sets shall be arithmetized in base-7 notation[1] as follows: a set $S \subseteq \mathbb{N}$ is in Σ^0_k if it is representable as

$$S = \{\, (w)_7 \mid \exists x_1 \in \{\mathtt{3}, \mathtt{6}\}^* \forall x_2 \in \{\mathtt{3}, \mathtt{6}\}^* \ldots Q_k x_k \in \{\mathtt{3}, \mathtt{6}\}^* (\mathtt{1} x_1 \mathtt{1} y_1 \mathtt{1} \ldots x_k \mathtt{1} y_k \mathtt{1} w)_7 \in R\},$$

for some recursive set $R \subseteq \mathbb{N}$, where $(w)_7$ for $w \in \{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{6}\}^*$ denotes the natural number with base-7 notation w. The strings $x_i \in \{\mathtt{3}, \mathtt{6}\}^*$ represent *binary* notation of some numbers,

[1] Base 7 is the smallest base, for which the details of the constructions could be conveniently implemented.

where $\mathtt{3}$ stands for zero and $\mathtt{6}$ stands for one. The notation $(x)_2$ for $x \in \{\mathtt{3}, \mathtt{6}\}^*$ shall be used to denote the number represented by this encoding. The digits $\mathtt{1}$ act as separators. Throughout this paper, the set of base-7 digits $\{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{6}\}$ shall be denoted by Ω_7.

In general, the construction of a system of equations representing the set S begins with representing R, and proceeds with evaluating the quantifiers, eliminating the prefixes $\mathtt{1}x_1$, $\mathtt{1}x_2$, and so on until $\mathtt{1}x_k$. In the end, all numbers $(\mathtt{1}w)_7$ with $(w)_7 \in S$ will be produced. These manipulations can be expressed in terms of the following three functions:

$$\begin{aligned} \mathit{Remove}_{\mathtt{1}}(X) &= \{(w)_7 \mid (\mathtt{1}w)_7 \in X\}, \\ E(X) &= \{(\mathtt{1}w)_7 \mid \exists x \in \{\mathtt{3}, \mathtt{6}\}^* :\ (x\mathtt{1}w)_7 \in X\}, \\ A(X) &= \{(\mathtt{1}w)_7 \mid \forall x \in \{\mathtt{3}, \mathtt{6}\}^* :\ (x\mathtt{1}w)_7 \in X\}. \end{aligned}$$

The expression converting numbers of the form $(\mathtt{1}w)_7$ to $(w)_7$ is constructed as follows:

Lemma 5 (Removing leading digit $\mathtt{1}$)**.** *The value of the expression*

$$(X - \{1\} \cap \{0\}) \cup \bigcup_{i \in \Omega_7 \setminus \{\mathtt{0}\}} \bigcup_{t \in \{0,1\}} \big[(X \cap (\mathtt{1}i\Omega_7^t(\Omega_7^2)^*)_7) \dot{-} (\mathtt{10}^*)_7\big] \cap (i\Omega_7^t(\Omega_7^2)^*)_7 \tag{3.1}$$

on any $S \subseteq (\mathtt{1}(\Omega_7^* \setminus \mathtt{0}\Omega_7^*))_7$ *is* $\{(w)_7 \mid (\mathtt{1}w)_7 \in S\}$. *The value on* $S \subseteq (\mathtt{10}\Omega_7^*)_7$ *equals* $\varnothing$.

With Lemma 5 established and the expression (3.1) proved to implement the function $\mathit{Remove}_{\mathtt{1}}(X)$, the notation $\mathit{Remove}_{\mathtt{1}}(X)$ is used in equations to refer to this subexpression.

Next, consider the function $E(X)$ representing the existential quantifier ranging over strings in $\{\mathtt{3}, \mathtt{6}\}^*$. This function can be implemented by a single expression as follows:

Lemma E (Representing the existential quantifier)**.** *The value of the expression*

$$(X \cap (\mathtt{1}\Omega_7^*)_7) \cup \Big(\big[(X \cap (\{\mathtt{3}, \mathtt{6}\}^+\mathtt{1}\Omega_7^*)_7) \dot{-} (\{\mathtt{3}, \mathtt{6}\}^+\mathtt{0}^*)_7\big] \cap (\mathtt{1}\Omega_7^*)_7\Big)$$

on any $S \subseteq (\{\mathtt{3}, \mathtt{6}\}^*\mathtt{1}\Omega_7^*)_7$ *is* $E(S) = \{(\mathtt{1}w)_7 \mid \exists w' \in \{\mathtt{3}, \mathtt{6}\}^*(w'\mathtt{1}w)_7 \in S\}$.

Note that $E(X)$ can already produce any recursively enumerable set from a recursive argument, and therefore it is essential to use subtraction in the expression.

With the existential quantifier implemented, the next task is to represent a universal quantifier. Ideally, one would be looking for an expression implementing $A(X)$, but, unfortunately, no such expression was found, and the actual construction given below implements the universal quantifier using multiple equations. The first step is devising an equation representing the function $f(X) = \{(x\mathtt{1}w)_7 \mid x \in \{\mathtt{3}, \mathtt{6}\}^*,\ (\mathtt{1}w)_7 \in X\}$, which appends every string of digits in $\{\mathtt{3}, \mathtt{6}\}^*$ to numbers in its argument set.

Lemma 6. *For every constant set* $X \subseteq (\mathtt{1}\Omega_7^*)_7$, *the equation*

$$Y = X \cup \mathit{Append}_{\mathtt{3},\mathtt{6}}(Y), \quad \textit{where}$$

$$\begin{aligned} \mathit{Append}_{\mathtt{3},\mathtt{6}}(Y) = \bigcup_{i,j \in \{\mathtt{3},\mathtt{6}\}} &\Big[\Big(\big[(Y \cap (j\Omega_7^*)_7) + (\mathtt{20}^*)_7\big] \cap (\mathtt{2}j\Omega_7^*)_7\Big) + ((i-2)\mathtt{0}^*)_7\Big] \cap (ij\Omega_7^*)_7 \\ \cup \bigcup_{i \in \{\mathtt{3},\mathtt{6}\}} &\big[(Y \cap (\mathtt{1}\Omega_7^*)_7) + (i\mathtt{0}^*)_7\big] \cap (i\mathtt{1}\Omega_7^*)_7 \end{aligned}$$

has the unique solution $Y = \{(x\mathtt{1}w)_7 \mid x \in \{\mathtt{3}, \mathtt{6}\}^*, (\mathtt{1}w)_7 \in X\}$.

Lemma A (Representing the universal quantifier)**.** *Let $S, \widetilde{S} \subseteq (\{\mathtt{3},\mathtt{6}\}^*\mathtt{1}\Omega_7^*)_7$ be any sets, such that $\widetilde{S} \cap S = \varnothing$ and for $x', x \in \{\mathtt{3},\mathtt{6}\}^*$ $(x\mathtt{1}w)_7 \in S$ and $(x'\mathtt{1}w)_7 \notin S$ implies $(x'\mathtt{1}w)_7 \in \widetilde{S}$. Then the following system of equations over sets of integers in variables Y, $\widetilde{Y}$ and Z*

$$Y = Z \cup \mathit{Append}_{\mathtt{3},\mathtt{6}}(Y)$$

$$\widetilde{Y} = E(\widetilde{S}) \cup \mathit{Append}_{\mathtt{3},\mathtt{6}}(\widetilde{Y})$$

$$Z \subseteq (\mathtt{1}\Omega_7^+)_7$$

$$Y \subseteq S \subseteq Y \cup \widetilde{Y},$$

has the unique solution $Z = A(S) = \{(\mathtt{1}w)_7 \,|\, \forall x \in \{\mathtt{3},\mathtt{6}\}^* : (x\mathtt{1}w)_7 \in S\}$, $Y = \{(y\mathtt{1}w)_7 \,|\, y \in \{\mathtt{3},\mathtt{6}\}^*, \forall x \in \{\mathtt{3},\mathtt{6}\}^* : (x\mathtt{1}w)_7 \in S\}$, $\widetilde{Y} = \{(y\mathtt{1}w)_7 \,|\, y \in \{\mathtt{3},\mathtt{6}\}^*, \exists x \in \{\mathtt{3},\mathtt{6}\}^* : (x\mathtt{1}w)_7 \in \widetilde{S}\}$.

Once the above quantifiers process a number $(\mathtt{1}x_k\mathtt{1}x_{k-1}\ldots\mathtt{1}x_1\mathtt{1}w)_7$, reducing it to $(\mathtt{1}w)_7$, the actual number $(w)_7$ is obtained from this encoding by Lemma 5.

Theorem 1. *Every arithmetical set $S \subseteq \mathbb{Z}$ ($S \subseteq \mathbb{N}$) is representable as a component of a unique solution of a system of equations over sets of integers (sets of natural numbers, respectively) with φ_j, ψ_j using the operations of addition and union and ultimately periodic constants (addition, subtraction, union and singleton constants, respectively).*

4. Representing hyper-arithmetical sets

Following Moschovakis [14, Sec. 8E] and Aczel [1, Thm. 2.2.3], *hyper-arithmetical* sets $B_1, B_2, \ldots$ shall be defined as the *smallest effective σ-ring*, which is the recursion-theoretic counterpart to Borel sets (the smallest family of sets containing all open sets and closed under countable union and countable intersection).

Let $f_1, f_2, \ldots$ be an enumeration of all partial recursive functions and let τ_1, τ_2 be two recursive functions. Then, for all $k \in \mathbb{N}$,

$$B_{\tau_1(k)} = \mathbb{N} \setminus \{k\}, \quad C_{\tau_1(k)} = \{k\}$$

Moreover, for all numbers $k \in \mathbb{N}$, if f_k is a total function, then

$$B_{\tau_2(k)} = \bigcup_{n \in \mathbb{N}} C_{f_k(n)}, \quad C_{\tau_2(k)} = \bigcap_{n \in \mathbb{N}} B_{f_k(n)},$$

where the former operation is known as *effective σ-union*, while the latter is *effective σ-intersection.* Note that the only distinction between B_e and C_e is that the former is defined as a union and the latter as an intersection. As the definitions are dual, $B_e = \overline{C_e}$.

The family of sets $\mathcal{B} = \{B_e, C_e \mid e \in I\}$, where $I \subseteq \mathbb{N}$ is an index set, is called an effective σ-ring, if it contains $\{B_{\tau_1(e)}, C_{\tau_1(e)} \mid e \in \mathbb{N}\}$ and is closed under effective σ-union and effective σ-intersection. Then the hyper-arithmetical sets are defined as the smallest effective σ-ring, which can be formally defined as the least fixed point of a certain operator on the set $\mathcal{A} = 2^{\mathbb{N} \times 2^{\mathbb{N}} \times 2^{\mathbb{N}}}$, where a triple (e, B_e, C_e) indicates that the sets B_e and C_e have been defined for the index e in the above inductive definition, and an operator $\Phi : \mathcal{A} \to \mathcal{A}$ represents one step of this inductive definition. Furthermore, this least fixed point can be obtained constructively by a transfinite induction on countable ordinals, which is essential for any proofs about hyper-arithmetical sets. It is known [14, Sec. 8E] [1, Thm. 2.2.3] that for some (easy) choices of τ_1 and τ_2 the smallest effective σ-ring coincides with Δ_1^1 sets.

Fix those two functions and the corresponding $\mathcal{B}$. Note that the definition is valid not for every choice of τ_1 and τ_2: in particular, they must be one-to-one and have disjoint images.

With every set $B_e \in \mathcal{B}$ one can associate a *tree of* B_e, labelled with sets from $\mathcal{B}$: its root is labelled with B_e, and each vertex $B_{\tau_2(e')}$ ($C_{\tau_2(e')}$, respectively) in the tree has children labelled with $\{C_{f_{e'}(n)} \mid n \in \mathbb{N}\}$ ($\{B_{f_{e'}(n)} \mid n \in \mathbb{N}\}$, respectively). Vertices of the form $B_{\tau_1(e')}$ or $C_{\tau_1(e')}$ have no children; these are the only leaves in the tree.

A partial order $\prec$ is *well-founded*, if it has no infinite descending chain. Extending this notion to oriented trees, a tree is well-founded if it contains no infinite downward path.

Lemma 7. *For each pair of sets $B_e, C_e \in \mathcal{B}$ the trees of B_e, C_e are well-founded.*

The well-foundedness of a set allows using the *well-founded induction principle*: given a property ϕ and a well founded order $\prec$ on a set A, $\phi(n)$ is true for all $n \in A$ if

$$(\forall m \prec n \; \phi(m)) \Rightarrow \phi(n).$$

This principle shall be used in the proof of the main construction, which is described in the rest of this section. Note, that the basis of the induction are $\prec$-minimal elements n of A, as for them $\phi(n)$ has to be shown directly.

Fix B_{i_0} as the target set in the root. Consider a path of length k in this tree, going from B_{i_0} to C_{i_1}, B_{i_2}, …, B_{i_k} (or C_{i_k}, depending on the parity of k). Then, for each j-th set in this path, $i_j = f_{\tau_2^{-1}(i_{j-1})}(n_j)$ for some number n_j, and the path is uniquely defined by the sequence of numbers $n_1, \dots, n_k$. Consider the binary encoding of each of these numbers written using digits $\mathtt{3}$ and $\mathtt{6}$ (representing zero and one, respectively), and let *Resolve* be a partial function that maps finite sequences of such "binary" strings representing numbers $n_1, \dots, n_k$ to the number i_k of the set B_{i_k} or C_{i_k} in the end of this path. The value of this function can be formally defined by induction:

$$\mathit{Resolve}(\langle\rangle) = i_0, \qquad \mathit{Resolve}(x_1, \dots, x_k) = f_{\tau_2^{-1}(\mathit{Resolve}(x_1,\dots,x_{k-1}))}((x_k)_2),$$

Note that *Resolve* may be undefined if some τ_2-preimage is undefined.

The goal is to construct a system of equations, such that the following two sets are among the components of its unique solution:

$$\mathit{Goal}_0 = \{(\mathtt{1}x_k\mathtt{1}x_{k-1}\dots\mathtt{1}x_1\mathtt{10}w)_7 \mid k \geqslant 0, x_i \in \{\mathtt{3},\mathtt{6}\}^*, (w)_7 \in B_{\mathit{Resolve}(x_1,\dots,x_k)}\},$$
$$\mathit{Goal}_1 = \{(\mathtt{1}x_k\mathtt{1}x_{k-1}\dots\mathtt{1}x_1\mathtt{10}w)_7 \mid k \geqslant 0, x_i \in \{\mathtt{3},\mathtt{6}\}^*, (w)_7 \in C_{\mathit{Resolve}(x_1,\dots,x_k)}\}.$$

These sets encode the sets $B_0, B_1, \dots$ needed to compute B_{i_0}. In this way the (possibly infinite) amount of equations defining sets in hyper-arithmetical hierarchy is encoded in a finite amount of equations using only small number of variables. The set B_i in the node with path to the root encoded by $x_k, x_{k-1}, \dots, x_1 \in \{\mathtt{3},\mathtt{6}\}^*$ is represented by $\{(\mathtt{1}x_k\mathtt{1}\dots\mathtt{1}x_k\mathtt{10}w)_7 \mid (w)_7 \in B_i\} \subseteq \mathit{Goal}_0$.

The following set defines the admissible encodings, that is, numbers encoding paths in the tree of B_{i_0}:

$$\mathit{Admissible} = \{(\mathtt{1}x_k\mathtt{1}x_{k-1}\mathtt{1}\dots\mathtt{1}x_1\mathtt{10}w)_7 | k \geqslant 0, x_i \in \{\mathtt{3},\mathtt{6}\}^*, \mathit{Resolve}(x_1,\dots,x_k) \text{ is defined}\}$$

The next two sets represent the leaves of the tree of B_{i_0}, and the numbers in those leaves:

$$\begin{aligned} R_0 = \{&(\mathtt{1}x_k\mathtt{1}x_{k-1}\dots\mathtt{1}x_1\mathtt{10}w)_7 \mid \\ &\quad k \geqslant 0,\ x_i \in \{\mathtt{3},\mathtt{6}\}^*, \exists e \in \mathbb{N} : \mathit{Resolve}(x_1,\dots,x_k) = \tau_1(e),\ (w)_7 \in B_{\tau_1(e)}\}, \end{aligned}$$

$$R_1 = \{(\mathtt{1}x_k\mathtt{1}x_{k-1}\dots\mathtt{1}x_1\mathtt{10}w)_7 \mid \\ k \geqslant 0,\ x_i \in \{\mathtt{3},\mathtt{6}\}^*, \exists e \in \mathbb{N} : \mathit{Resolve}(x_1,\dots,x_k) = \tau_1(e),\ (w)_7 \in C_{\tau_1(e)}\}.$$

Lemma 8. *The sets Goal_i, $\mathit{Admissible}$, R_i are r.e. sets, $\mathit{Resolve}$ is an r.e. predicate.*

Consider the following system of equations:

$$X_0 = E(\mathit{Remove}_{\mathtt{1}}(X_1)) \cup R_0 \tag{4.1}$$

$$X_1 = Z \cup R_1 \tag{4.2}$$

$$\widetilde{Y} = E(\mathit{Remove}_{\mathtt{1}}(X_1)) \cup \mathit{Append}_{\mathtt{3},\mathtt{6}}(\widetilde{Y}) \tag{4.3}$$

$$Y = Z \cup \mathit{Append}_{\mathtt{3},\mathtt{6}}(Y) \tag{4.4}$$

$$Y \subseteq \mathit{Remove}_{\mathtt{1}}(X_0 \cap \mathit{Admissible}) \subseteq Y \cup \widetilde{Y} \tag{4.5}$$

$$Z \subseteq (\mathtt{1}\Omega_7^+)_7 \tag{4.6}$$

$$X_0, X_1 \subseteq \mathit{Admissible} \tag{4.7}$$

$$X_0 \cap R_1 = X_1 \cap R_0 = \varnothing \tag{4.8}$$

Its intended unique solution has $X_0 = \mathit{Goal}_0$ and $X_1 = \mathit{Goal}_1$, and accordingly encodes the set B_{i_0}, as well as all sets of $\mathcal{B}$ on which B_{i_0} logically depends. The system implements the functions $E(X)$ and $A(X)$ to represent effective σ-union and σ-intersection, respectively. For that purpose, the expression for $E(X)$ introduced in Lemma E, as well as the system of equations implementing $A(X)$ defined in Lemma A, are applied iteratively to the same variables X_0 and X_1. Intuitively, the above system may be regarded as an implementation of an equation $X_0 = A(E(X_0)) \cup \mathit{const}$.

The proof uses the principle of induction on well-founded structures. The membership of numbers of the form $(\mathtt{1}x_k\mathtt{1}x_{k-1}\dots\mathtt{1}x_1\mathtt{10}w)_7$ in the variables X_0 and X_1, where $k \geqslant 0$, $x_i \in \{\mathtt{3},\mathtt{6}\}^*$ and $w \in \Omega_7^* \setminus \mathtt{0}\Omega_7^*$, is first proved for larger k's and then inductively extended down to $k = 0$, which allows extracting B_{i_0} out of the solution. The well-foundedness of the tree of B_{i_0} means that although B_{i_0} depends upon infinitely many sets, each dependency is over a finite path ending with a constant, that is, the self-dependence of numbers in X_0, X_1 on the numbers in X_0, X_1 reaches a constant R_0, R_1 in finitely many steps (yet the number of steps is unbounded).

Lemma 9. *The unique solution of the system* (4.1)–(4.8) *is*

$$X_0 = \mathit{Goal}_0 = \{(\mathtt{1}x_k\dots\mathtt{1}x_1\mathtt{10}w)_7 \mid k \geqslant 0, x_i \in \{\mathtt{3},\mathtt{6}\}^*, (w)_7 \in B_{\mathit{Resolve}(x_1,\dots,x_k)}\}$$

$$X_1 = \mathit{Goal}_1 = \{(\mathtt{1}x_k\dots\mathtt{1}x_1\mathtt{10}w)_7 \mid k \geqslant 0, x_i \in \{\mathtt{3},\mathtt{6}\}^*, (w)_7 \in C_{\mathit{Resolve}(x_1,\dots,x_k)}\}$$

$$Y = \{(x_{k+1}\mathtt{1}x_k\dots\mathtt{1}x_1\mathtt{10}w)_7 \mid k \geqslant 0, x_i \in \{\mathtt{3},\mathtt{6}\}^*, \forall x_{k+1} : (w)_7 \in B_{\mathit{Resolve}(x_1,\dots,x_{k+1})}\}$$

$$\widetilde{Y} = \{(x_{k+1}\mathtt{1}x_k\dots\mathtt{1}x_1\mathtt{10}w)_7 \mid k \geqslant 0, x_i \in \{\mathtt{3},\mathtt{6}\}^*, \exists x_{k+1} : (w)_7 \in C_{\mathit{Resolve}(x_1,\dots,x_{k+1})}\}$$

$$Z = \mathit{Goal}_1 \setminus R_1 = \{(\mathtt{1}x_k\dots\mathtt{1}x_1\mathtt{10}w)_7 \mid \\ k \geqslant 0, e \in \mathbb{N}, x_i \in \{\mathtt{3},\mathtt{6}\}^*, \mathit{Resolve}(x_1,\dots,x_k) = \tau_2(e), (w)_7 \in C_{\tau_2(e)}\}$$

Then, in order to obtain the set B_{i_0}, it remains to intersect $X_0 = \mathit{Goal}_0$ with the recursive constant set $(\mathtt{10}\Omega_7^*)_7$, and then remove the leading digits $\mathtt{10}$ by a construction analogous to the one in Lemma 5.

Theorem 2. *For every hyper-arithmetical set $B \subseteq \mathbb{Z}$ ($B \subseteq \mathbb{N}$) there is a system of equations over subsets of $\mathbb{Z}$ (over subsets of $\mathbb{N}$, respectively) using union, addition and ultimately* *periodic constants (union, addition, subtraction and singleton constants, respectively), such that $(B, \dots)$ is its unique solution.*

5. Equations with addition only

Equations over sets of natural numbers with addition as the only operation can represent an *encoding* of every recursive set, with each number $n \in \mathbb{N}$ represented by the number $16n + 13$ in the encoding [9]. In order to define this encoding, for each $i \in \{0, 1, \dots, 15\}$ and for every set $S \subseteq \mathbb{Z}$, denote:

$$\tau_i(S) = \{16n + i \mid n \in S\}.$$

The encoding of a set of natural numbers $\widehat{S} \subseteq \mathbb{N}$ is defined as

$$S = \sigma_0(\widehat{S}) = \{0\} \cup \tau_6(\mathbb{N}) \cup \tau_8(\mathbb{N}) \cup \tau_9(\mathbb{N}) \cup \tau_{12}(\mathbb{N}) \cup \tau_{13}(\widehat{S}),$$

Proposition 2 ([9, Thm. 5.3]). *For every recursive set S there exists a system of equations over sets of natural numbers in variables $X, Y_1, \dots, Y_m$ using the operation of addition and ultimately periodic constants, which has a unique solution with $X = \sigma_0(S)$.*

This result is proved by first representing the set S by a system with addition and union, and then by representing addition and union of sets using addition of their σ_0-encodings.

The purpose of this section is to obtain a similar result for equations over sets of integers: namely, that they can represent the same kind of encoding of every hyper-arithmetical set. For every set $\widehat{S} \subseteq \mathbb{Z}$, define its *encoding* as the set

$$S = \sigma(\widehat{S}) = \{0\} \cup \tau_6(\mathbb{Z}) \cup \tau_8(\mathbb{Z}) \cup \tau_9(\mathbb{Z}) \cup \tau_{12}(\mathbb{Z}) \cup \tau_{13}(\widehat{S}).$$

The subset $S \cap \{16n + i \mid n \in \mathbb{Z}\}$ is called the *i-th track of S*.

The first result on this encoding is that the condition of a set X being an encoding of any set can be specified by an equation of the form $X + C = D$.

Lemma 10 (cf. [9, Lemma 3.3]). *A set $X \subseteq \mathbb{Z}$ satisfies an equation*

$$X + \{0, 4, 11\} = \bigcup_{\substack{i \in \{0,1,3,4,6,7,\\ 8,9,10,12,13\}}} \tau_i(\mathbb{Z}) \cup \{11\}$$

if and only if $X = \sigma(\widehat{X})$ for some $\widehat{X} \subseteq \mathbb{Z}$.

Now, assuming that the given system of equations with union and addition is decomposed to have all equations of the form $X = Y + Z$, $X = Y \cup Z$ or $X = const$, these equations can be simulated in a new system as follows:

Lemma 11 (cf. [9, Lemma 4.1]). *For all sets $X, Y, Z \subseteq \mathbb{Z}$,*

$$\sigma(Y) + \sigma(Z) + \{0, 1\} = \sigma(X) + \sigma(\{0\}) + \{0, 1\} \quad \text{if and only if} \quad Y + Z = X$$
$$\sigma(Y) + \sigma(Z) + \{0, 2\} = \sigma(X) + \sigma(X) + \{0, 2\} \quad \text{if and only if} \quad Y \cup Z = X.$$

Using these two lemmata, one can simulate any system with addition and union by a system with addition only. Taking systems representing different hyper-arithmetical sets, the following result on the expressive power of systems with addition can be established:

	Sets representable by unique solutions	Complexity of decision problems	
		solution existence	solution uniqueness
over $2^{\mathbb{N}}$, with $\{+,\cup\}$	Δ^0_1 (recursive) [8]	Π^0_1-complete [8]	Π^0_2-complete [8]
over $2^{\mathbb{N}}$, with $\{+\}$	encodings of Δ^0_1 [9]	Π^0_1-complete [9]	Π^0_2-complete [9]
over $2^{\mathbb{N}}$, with $\{+,\dot{-},\cup\}$	Δ^1_1 (hyper-arithmetical)	Σ^1_1-complete	$\Pi^1_1 \leqslant \cdot \leqslant \Delta^1_2$
over $2^{\mathbb{Z}}$, with $\{+,\cup\}$	Δ^1_1	Σ^1_1-complete	$\Pi^1_1 \leqslant \cdot \leqslant \Delta^1_2$
over $2^{\mathbb{Z}}$, with $\{+\}$	encodings of Δ^1_1	Σ^1_1-complete	$\Pi^1_1 \leqslant \cdot \leqslant \Delta^1_2$

Table 1: Summary of the results.

Theorem 3. *For every hyper-arithmetical set $S \subseteq \mathbb{Z}$ there exists a system of equations over sets of integers using the operation of addition and ultimately periodic constants, which has a unique solution with $X_1 = T$, where $S = \{n \mid 16n \in T\}$.*

6. Decision problems

Having a solution (solution existence) and having exactly one solution (solution uniqueness) are basic properties of a system of equations. For language equations with continuous operations, *solution existence* is Π^0_1-complete [19], and it remains Π^0_1-complete already in the case of a unary alphabet, concatenation as the only operation and regular constants [9], that is, for equations over sets of natural numbers with addition only. For the same formalisms, *solution uniqueness* is Π^0_2-complete.

Consider equations over sets of integers. Since their expressive power extends beyond the arithmetical hierarchy, the decision problems should accordingly be harder. In fact, the solution existence is Σ^1_1-complete, which will now be proved using a reduction from the following problem:

Proposition 3 (Rogers [21, Thm. 16-XX])**.** *Consider trees with nodes labelled by finite sequences of natural numbers, such that a node $(x_1, \dots, x_{k-1}, x_k)$ is a son of $(x_1, \dots, x_{k-1})$, and the empty sequence ε is the root. Then the following problem is Π^1_1-complete: "Given a description of a Turing machine recognizing the set of nodes of a certain tree, determine whether this tree has no infinite paths".*

In other words, a given Turing machine recognizes sequences of natural numbers, and the task is to determine whether there is *no* infinite sequence of natural numbers, such that all of its prefixes are accepted by the machine. The Σ^1_1-complete complement of the problem is testing whether such an infinite sequence exists, and it can be reformulated as follows:

Corollary 1. *The following problem is Σ^1_1-complete: "Given a Turing machine M working on natural numbers, determine whether there exists an infinite sequence of strings $\{x_i\}_{i=1}^{\infty}$ with $x_i \in \{\mathtt{3},\mathtt{6}\}^*$, such that M accepts $(\mathtt{1}x_k\mathtt{1}x_{k-1}\dots\mathtt{1}x_1\mathtt{1})_7$ for all $k \geqslant 0$".*

This problem can be reduced to testing existence of a solution of equations over sets of numbers.

Theorem 4. *The problem of whether a given system of equations over sets of integers with addition and ultimately periodic constants has a solution is Σ^1_1-complete.*

Now consider the solution uniqueness property. The following upper bound on its complexity naturally follows by definition:

Theorem 5. *The problem of whether a given system of equations over sets of integers using addition and ultimately periodic constants has a unique solution can be represented as a conjunction of a Σ^1_1-formula and a Π^1_1-formula, and is accordingly in Δ^1_2. At the same time, the problem is Π^1_1-hard.*

The exact hardness of testing solution uniqueness is still open. The properties of different families of equations over sets of numbers are summarized in Table 1.

References

[1] P. Aczel, "An introduction to inductive definitions", in: J. Barwise (Ed.), *Handbook of Mathematical Logic*, 739–783, North-Holland, 1977.

[2] F. d'Alessandro, J. Sakarovitch, "The finite power property in free groups", *Theoretical Computer Science*, 293:1 (2003), 55–82.

[3] A. V. Anisimov, "Languages over free groups", *Mathematical Foundations of Computer Science*, (MFCS 1975, Mariánské Lázně, September 1–5, 1975), LNCS 32, 167–171.

[4] S. Ginsburg, H. Rice, "Two families of languages related to ALGOL", *J. of the ACM*, 9 (1962), 350–371.

[5] J. Y. Halpern, "Presburger arithmetic with unary predicates is Π^1_1 complete", *Journal of Symbolic Logic*, 56:2 (1991), 637–642.

[6] A. Jeż, "Conjunctive grammars can generate non-regular unary languages", *International Journal of Foundations of Computer Science*, 19:3 (2008), 597–615.

[7] A. Jeż, A. Okhotin, "Conjunctive grammars over a unary alphabet: undecidability and unbounded growth", *Theory of Computing Systems*, 46:1 (2010), 27–58.

[8] A. Jeż, A. Okhotin, "On the computational completeness of equations over sets of natural numbers" *ICALP 2008* (Reykjavik, Iceland, July 7–11, 2008), LNCS 5126, 63–74.

[9] A. Jeż, A. Okhotin, "Equations over sets of natural numbers with addition only", *STACS 2009* (Freiburg, Germany, 26–28 February, 2009), 577–588.

[10] M. Kunc, "The power of commuting with finite sets of words", *Theory of Computing Systems*, 40:4 (2007), 521–551.

[11] M. Kunc, "What do we know about language equations?", *Developments in Language Theory* (DLT 2007, Turku, Finland, July 3–6, 2007), LNCS 4588, 23–27.

[12] T. Lehtinen, A. Okhotin, "On equations over sets of numbers and their limitations", *Developments in Language Theory* (DLT 2009, Stuttgart, Germany, 30 June–3 July, 2009), LNCS 5583, 360–371.

[13] P. McKenzie, K. Wagner, "The complexity of membership problems for circuits over sets of natural numbers", *Computational Complexity*, 16:3 (2007), 211–244.

[14] Y. Moschovakis, *Elementary Induction on Abstract Structures*, North-Holland, 1974.

[15] A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.

[16] A. Okhotin, "Conjunctive grammars and systems of language equations", *Programming and Computer Software*, 28:5 (2002), 243–249.

[17] A. Okhotin, "Unresolved systems of language equations: expressive power and decision problems", *Theoretical Computer Science*, 349:3 (2005), 283–308.

[18] A. Okhotin, "Computational universality in one-variable language equations", *Fundamenta Informaticae*, 74:4 (2006), 563–578.

[19] A. Okhotin, "Decision problems for language equations", *Journal of Computer and System Sciences*, 76 (2010), to appear; earlier version at ICALP 2003.

[20] J. Robinson, "An introduction to hyperarithmetical functions", *Journal of Symbolic Logic*, 32:3 (1967), 325–342.

[21] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967.

[22] S. D. Travers, "The complexity of membership problems for circuits over sets of integers" *Theoretical Computer Science*, 369:1–3 (2006), 211–229.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 489-500
www.stacs-conf.org

A $\frac{4}{3}$-COMPETITIVE RANDOMIZED ALGORITHM FOR ONLINE SCHEDULING OF PACKETS WITH AGREEABLE DEADLINES

ŁUKASZ JEŻ [1]

University of Wrocław, Institute of Computer Science
E-mail address: lje@cs.uni.wroc.pl
URL: http://www.ii.uni.wroc.pl/ lje/

Abstract. In 2005 Li et al. gave a ϕ-competitive deterministic online algorithm for scheduling of packets with agreeable deadlines [12] with a very interesting analysis. This is known to be optimal due to a lower bound by Hajek [7]. We claim that the algorithm by Li et al. can be slightly simplified, while retaining its competitive ratio. Then we introduce randomness to the modified algorithm and argue that the competitive ratio against oblivious adversary is at most $\frac{4}{3}$. Note that this still leaves a gap between the best known lower bound of $\frac{5}{4}$ by Chin et al. [5] for randomized algorithms against oblivious adversary.

1. Introduction

We consider the problem of *buffer management with bounded delay* (aka *packet scheduling*), introduced by Kesselman et al. [11]. It models the behaviour of a single network switch. We assume that time is slotted and divided into steps. At the beginning of a time step, any number of packets may arrive at a switch and are stored in its *buffer*. A packet has a positive weight and a deadline, which is the number of step right before which the packet expires: unless it has already been transmitted, it is removed from the buffer at the very beginning of that step and thus can no longer be transmitted. Only one packet can be transmitted in a single step. The goal is to maximize the *weighted throughput*, i.e., the total weight of transmitted packets.

As the process of managing packet queue is inherently a real-time task, we investigate the online variant of the problem. This means that the algorithm has to base its decision of which packet to transmit solely on the packets which have already arrived at a switch, without the knowledge of the future.

1998 ACM Subject Classification: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems.

Key words and phrases: online algorithms, scheduling, buffer management.

This work has been supported by MNiSW grants no. N N206 1723 33, 2007–2010 and N N206 490638, 2010–2011.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2479

1.1. Competitive Analysis.

To measure the performance of an online algorithm, we use a standard notion of competitive analysis [3], which compares the gain of the algorithm to the gain of the optimal solution on the same input sequence. For any algorithm Alg, we denote its gain on the input sequence I by $\mathcal{G}_{\text{ALG}}(I)$; we denote the optimal offline algorithm by Opt. We say that a deterministic algorithm Alg is $\mathcal{R}$-competitive if on any input sequence I, it holds that $\mathcal{G}_{\text{ALG}}(I) \geq \frac{1}{\mathcal{R}} \cdot \mathcal{G}_{\text{OPT}}(I)$.

When analysing the performance of an online algorithm Alg, we view the process as a game between Alg and an *adversary.* The adversary controls the packets' injection into the buffer and chooses which of them to send. The goal is then to show that the adversary's gain is at most $\mathcal{R}$ times Alg's gain.

If the algorithm is randomized, we consider its expected gain, $\mathbf{E}[\mathcal{G}_{\text{ALG}}(I)]$, where the expectation is taken over all possible random choices made by Alg. However, in the randomized case, the power of the adversary has to be further specified. Following Ben-David et al. [1], we distinguish between an *oblivious* and *adaptive-online* adversary (called adaptive for short). An oblivious adversary has to construct the whole input sequence in advance, not knowing the random bits used by an algorithm. The expected gain of Alg is compared to the gain of the optimal offline solution on I. An adaptive adversary decides packet injections upon seeing which packets are transmitted by the algorithm. However, it has to provide an answering entity Adv, which creates a solution on-line (in parallel to Alg) and cannot change it afterwards. We say that Alg is $\mathcal{R}$-competitive against an adaptive adversary if for any input sequence I created adaptively and any answering algorithm Adv, it holds that $\mathbf{E}[\mathcal{G}_{\text{ALG}}(I)] \geq \frac{1}{\mathcal{R}} \cdot \mathbf{E}[\mathcal{G}_{\text{ADV}}(I)]$. We note that Adv is (wlog) deterministic, but as Alg is randomized, so is the input sequence I.

In the literature on online algorithms (see e.g. [3]), the definition of the competitive ratio sometimes allows an additive constant, i.e., a deterministic algorithm is $\mathcal{R}$-competitive if there exists a constant $\alpha \geq 0$ such that $\mathcal{G}_{\text{ALG}}(I) \geq \frac{1}{\mathcal{R}} \cdot \mathcal{G}_{\text{OPT}}(I) - \alpha$ holds for evry input sequence I. An analogous definition applies to randomized case. Our upper bounds hold for $\alpha = 0$.

1.2. Previous work

The best known deterministic and randomized algorithms for general instances have competitive ratios at most $2\sqrt{2} - 1 \approx 1.828$ [6] and $e/(e-1) \approx 1.582$ [4], respectively. A recent analysis of the latter algorithm shows that it retains its competitive ratio even against adaptive-online adversary [8].

The best known lower bounds on competitive ratio against either adversary type use rather restricted 2-*bounded* sequences in which every packet has lifespan (deadline − release time) either 1 or 2. The lower bounds in question are $\phi \approx 1.618$ for deterministic algorithms [7], $\frac{4}{3}$ for randomized algorithms against adaptive adversary [2], and $\frac{5}{4}$ for randomized algorithms against oblivious adversary [5]. All these bounds are tight for 2-bounded sequences [11, 2, 4].

We restrict ourselves to sequences with *agreeable deadlines*, in which packets released later have deadlines at least as large as those released before ($r_i < r_j$ implies $d_i \leq d_j$). These strictly generalize the 2-bounded sequences. Sequences with agreeable deadlines also properly contain *s-uniform* sequences for all s, i.e., sequences in which every packet has lifespan exactly s. An optimal ϕ-competitive deterministic algorithm for sequences with agreeable deadlines is known [12].

Jeżabek studied the impact of *resource augmentation* on the deterministic competitive ratio [10, 9]. It turns out that while allowing the deterministic algorithm to transmit k packets in a single step for any constant k cannot make it 1-competitive (compared to the single-speed offline optimum) on unrestricted sequences [10], $k = 2$ is sufficient for sequences with agreeable deadlines [9].

1.3. Our contribution

Motivated by aforementioned results for sequences with agreeable deadlines, we investigate randomized algorithms for such instances. We devise a $\frac{4}{3}$-competitive randomized algorithm against oblivious adversary. The algorithm and its analysis are inspired by those by Li et al. [12] for deterministic case. The key insight is as follows. The algorithm MG by Li et al. [12] can be simplified by making it always send either e, the heaviest among the earliest non-dominated packets, or h, the earliest among the heaviest non-dominated packets. We call this algorithm MG$'$, and prove that it remains ϕ-competitive. Then we turn it into a randomized algorithm RG, simply by making it always transmit e with probability $\frac{w_e}{w_h}$ and h with the remaining probability. The proof of RG's $\frac{4}{3}$-competitiveness against oblivious adversary follows by similar analysis.

2. Preliminaries

We denote the release time, weight, and deadline of a packet j by r_j, w_j, and d_j, respectively. A packet j is *pending* at step t if $r_j \leq t$, it has not yet been transmitted, and $d_j > t$. We introduce a linear order $\trianglelefteq$ on the packets as follows: $i \trianglelefteq j$ if either

$$\begin{aligned} &d_i < d_j \ \text{, or} \\ &d_i = d_j \ \text{ and } w_i > w_j \ \text{, or} \\ &d_i = d_j \ \text{ and } w_i = w_j \ \text{ and } r_i \leq r_j \ . \end{aligned}$$

To make $\trianglelefteq$ truly linear we assume that in every single step the packets are released one after another rather then all at once, e.g. that they have unique fractional release times.

A *schedule* is a mapping from time steps to packets to be transmitted in those time steps. A schedule is *feasible* if it is injective and for every time step t the packet that t maps to is pending at t. It is convenient to view a feasible schedule S differently, for example as the set $\{S(t)\colon t > 0\}$, the sequence $S(1), S(2), \ldots$, or a matching in the *schedulability graph*. The schedulability graph is a bipartite graph, one of whose partition classes is the set of packets and the other is the set of time steps. Each packet j is connected precisely to each of the time steps t such that $r_j \leq t < d_j$ by an edge of weight w_j; an example is given in Figure 1. Observe that optimal offline schedules correspond to maximum weight matchings in the schedulability graph. Thus an optimal offline schedule can be found in polynomial time using the classic “Hungarian algorithm”, see for example [13]. One may have to remove appropriately chosen time step vertices first, so that the remaining ones match the number of packet vertices, though.

Given any linear order $\preceq$ on packets and a (feasible) schedule S, we say that S is consistent with $\preceq$, or that S is a $\preceq$-schedule, if for every t the packet $S(t)$ is the minimum pending packet with respect to $\preceq$. It is fairly easy to observe that if $\preceq$ is any *earliest deadline*

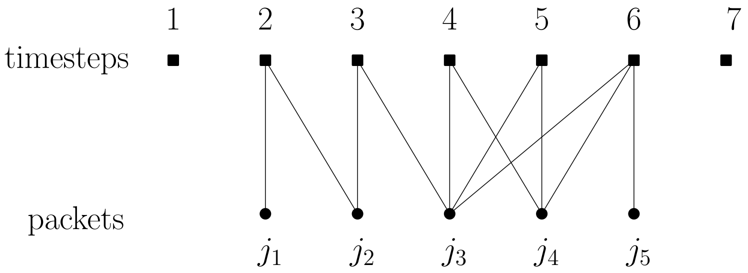


Figure 1: Schedulability graph for packets $j_1, j_2, \ldots, j_5$, whose release times and deadlines are (2,3), (2,4), (3,7), (4,7), (6,7) respectively; we ignore packet weights in the figure. Packets are represented by discs, time steps by squares.

first, with ties broken in an arbitrary way, then any feasible schedule can be turned to a unique $\preceq$-schedule by reordering its packets; in particular this applies to $\trianglelefteq$.

Recall that the *oblivious* adversary prepares the whole input sequence in advance and cannot alter it later on. Thus its solution is simply the offline optimal schedule for the complete sequence. Nevertheless, we still refer to the answering entity Adv rather than Opt in our analysis, as it involves altering the set of packets pending for the adversary, which may well be viewed as altering the input sequence. Now we introduce two schedules that are crucial for our algorithms and our analyzes.

Definition 2.1. The *oblivious schedule* at time step t, denoted O_t, is any fixed optimal feasible $\trianglelefteq$-schedule over all the packets pending at step t. For fixed O_t, a packet j pending at t is called *dominated* if $j \notin O_t$, and *non-dominated* otherwise. For fixed O_t let e denote $O_t(t)$, the $\trianglelefteq$-minimal of all non-dominated packets, and h denote the $\trianglelefteq$-minimal of all non-dominated maximum-weight packets.

Note that both the adversary and the algorithm can calculate their oblivious schedules at any step, and that these will coincide if their buffers are the same.

Definition 2.2. For a fixed input sequence, the *clairvoyant schedule* at time step t, denoted C_t, is any fixed optimal feasible schedule over all the packets pending at step t and all the packets that will arrive in the future.

Naturally, the adversary can calculate the clairvoyant schedule, as it knows the fixed input sequence, while the algorithm cannot, since it only knows the part of input revealed so far. However, the oblivious schedule gives some partial information about the clairvoyant schedule: intuitively, if p is dominated at t, it makes no sense to transmit it at t. Formally, (wlog) dominated packets are not included in the clairvoyant schedule, as stated in the following.

Fact 2.3. *For any fixed input sequence, time step t, and oblivious schedule O_t, there is a clairvoyant schedule C_t^* such that $C_t^* \cap \{j : r_j \leq t\} \subseteq O_t$.*

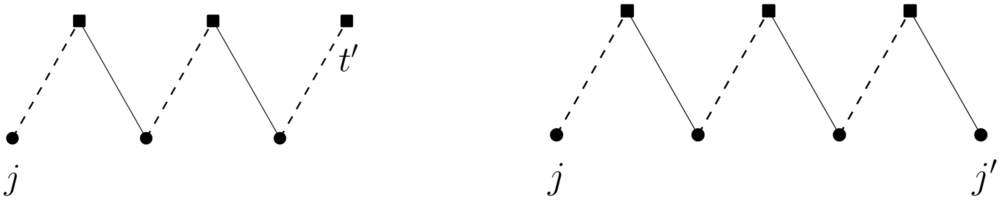
Proof. This is a standard alternating path argument about matchings. If you are unfamiliar with these concepts, refer to a book by A. Schrijver [13] for example.

Let O_t be the oblivious schedule and C_t be any clairvoyant schedule. Treat both as matchings in the schedulability graph and consider their symmetric difference $C_t \oplus O_t$.

Consider any job $j \in C_t \setminus O_t$ such that $r_j \leq t$. It is an endpoint of an alternating path P in $C_t \oplus O_t$. Note that all the jobs on P are already pending at time t: this is certainly true about j, and all the successive jobs belong to O_t, so they are pending as well.

First we prove that P has even length, i.e., it ends in a node corresponding to a job. Assume for contradiction that P's length is odd, and that P ends in a node corresponding to a timestep t'. Note that no job is assigned to t' in O_t. Then $O_t \oplus P$ is a feasible schedule that, treated as a set, satisfies $O_t \subseteq O_t \oplus P$ and $j \in O_t \oplus P$. This contradicts optimality of O_t. See Figure 2a for illustration.

Thus P has even length and ends with a job $j' \in O_t \setminus C_t$. By optimality of both O_t and C_t, $w_j = w_{j'}$ holds. Thus $C_t \oplus P$ is an optimal feasible schedule: in terms of sets the only difference between C_t and $C_t \oplus P$ is that j has been replaced by j', a job of the same weight. See Figure 2b for illustration.



(a) P cannot have odd length: in such case the assignment of jobs on P in O_t could be changed to match the strictly better assignment of C_t.

(b) P has even length: now the assignment of jobs on P in C_t can be changed to match the assignment of O_t so that the value of Δ drops.

Figure 2: The alternating path P. Packets are represented by discs, time steps by squares. Dashed lines represent C_t, solid lines represent O_t.

Applying such changes iteratively transforms C_t to a clairvoyant schedule C_t^* as announced. To observe that a finite number of iterations suffices, define $\Delta(S) := |S \cap \{j \colon r_j \leq t\} \setminus O_t|$ for any schedule S. It follows that $\Delta(C_t \oplus P) = \Delta(C_t) - 1$. Since Δ is non-negative and its value drops by one with each iteration, C_t^* is obtained in a finite number of steps. ∎

Definition 2.4. We say that a clairvoyant schedule C_t conforms with an oblivious schedule O_t if C_t is a $\trianglelefteq$-schedule, $C_t \cap \{j \colon r_j \leq t\} \subseteq O_t$, and for all $i \in O_t$ such that $i \triangleleft j = C_t(t)$, $w_i < w_j$ holds.

Fact 2.5. *For every oblivious schedule O_t there is a conforming clairvoyant schedule C_t^*.*

Proof. Let C_t be a clairvoyant schedule such that $C_t \cap \{j \colon r_j \leq t\} \subseteq O_t$; Fact 2.3 guarantees its existence. Let C_t^* be the schedule obtained from C_t by first turning it into a $\trianglelefteq$-schedule C'_t and then replacing $j = C'_t(t)$ with a $\trianglelefteq$-minimal non-dominated packet j' of the same weight.

If $j' = j$, then $C_t^* = C'_t$, and thus it is a clairvoyant $\trianglelefteq$-schedule. Assume $j' \neq j$, i.e., $j' \triangleleft j$. Then $j' \notin C'_t$, since C'_t is a $\trianglelefteq$-schedule. Thus C_t^* is feasible as we replace C'_t's very first packet by another pending packet which was not included in C_t. Observe that C_t^* is indeed a clairvoyant $\trianglelefteq$-schedule: optimality follows from $w_{j'} = w_j$, while consistency with $\trianglelefteq$ follows from $j' \triangleleft j$.

It remains to prove that for every $i \in O_t$ such that $i \lhd j'$, $w_i < w_{j'} = w_j$ holds. Note that $i \notin C_t^*$ as C_t^* is a $\unlhd$-schedule, and that $w_i \neq w_{j'} = w_j$ holds by the choice of j'. Assume for contradiction that $w_i > w_j$. Then C_t^* with j replaced by i is a feasible schedule contradicting optimality of C_t^*. ■

Now we inspect some properties of conforming schedules.

Fact 2.6. *Let C_t be a clairvoyant schedule conforming with an oblivious schedule O_t. If $i, j \in O_t$, $w_i < w_j$ and $d_i < d_j$ (or, equivalently $w_i < w_j$ and $i \lhd j$), and $i \in C_t$, then also $j \in C_t$.*

Proof. Assume for contradiction that $j \notin C_t$. Then C_t with i replaced by j is a feasible schedule contradicting optimality of C_t. ■

Lemma 2.7. *Let C_t be a clairvoyant schedule conforming with an oblivious schedule O_t. Suppose that $e = O_t(t) \notin C_t$. Then there is a clairvoyant schedule C_t^* obtained from C_t by reordering of packets such that $C_t^*(t) = h$.*

Proof. Let $j = C_t(t) \neq h$ and let $O_t = p_1, p_2, \ldots, p_s$. Observe that $h \in C_t$ by Fact 2.6. So in particular $e = p_1$, $j = p_k$, and $h = p_l$ for some $1 < k < l \leq s$. Let d_i denote the deadline of p_i for $1 \leq i \leq s$. Since O_t is feasible in the absence of future arrivals, $d_i \geq t + i$ for $i = 1, \ldots, s$.

Recall that $p_k, p_l \in C_t$ and that there can be some further packets $p \in C_t$ such that $p_k \lhd p \lhd p_l$; some of these packets may be not pending yet. We construct a schedule C_t' by reordering C_t. Precisely, we put all the packets from C_t that are not yet pending at t after all the packets from C_t that are already pending, keeping the order between the pending packets and between those not yet pending. By the agreeable deadlines property, this is an earliest deadline first order, so C_t' is a clairvoyant schedule.

As $e = p_1 \notin C_t'$ and $d_i \geq t + i$ for $i = 1, \ldots, s$, all the packets $x \in C_t'$ preceding h in C_t' (i.e., $x \in C_t'$ such that $r_x \leq t$ and $x \lhd p_l = h$) have *slack* in C_t', i.e., each of them could also be scheduled one step later. Hence $h = p_l$ can be moved to the very front of C_t' while keeping its feasibility, i.e., $C_t' = p_k, p_{k'}, \ldots, p_{l'}, p_l$ can be transformed to a clairvoyant schedule $C_t^* = p_l, p_k, p_{k'}, \ldots, p_{l'}$. The reordering is illustrated in Figure 3. ■

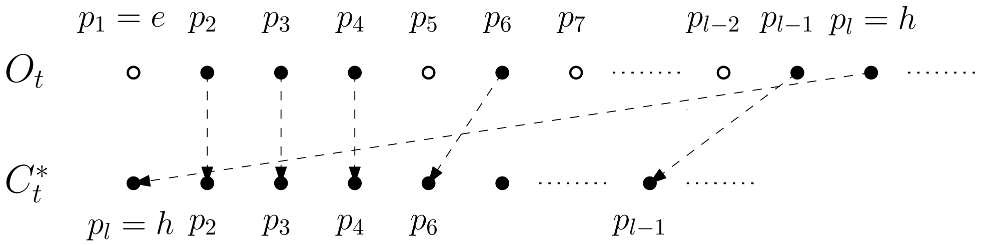


Figure 3: Construction of the schedule C_t^*. Packets are represented by circles: the ones included in C_t (C_t^*) are filled, the remaining ones are hollow.

3. Algorithms and their analyzes

3.1. The Algorithms

The algorithm MG [12] works as follows: at the beginning of each step t it considers the packets in the buffer and the newly arrived packets, and calculates O_t. Then MG identifies the packets e and h. If $\phi w_e \geq w_h$, MG sends e. Otherwise, it sends the $\trianglelefteq$-minimal packet f such that $w_f \geq \phi w_e$ and $\phi w_f \geq w_h$; the latter exists as h itself is a valid candidate. Our deterministic algorithm MG′ does exactly the same with one exception: if $\phi w_e < w_h$, it sends h rather than f. Our randomized algorithm RG also works in a similar fashion: it transmits e with probability $\frac{w_e}{w_h}$ and h with the remaining probability. For completeness, we provide pseudo-codes of all three algorithms in Figure 4.

```
MG (step $t$)
    $O_t \leftarrow$ oblivious schedule at $t$
    $e \leftarrow$ the $\trianglelefteq$-minimal packet from $O_t$
    $h \leftarrow$ the $\trianglelefteq$-minimal of all the heaviest packets from $O_t$
    if $\phi w_e \geq w_h$
        then transmit $e$
        else  $f \leftarrow$ the $\trianglelefteq$-minimal of all $j \in O_t$ s.t. $w_j \geq \phi w_e$ and $\phi w_j \geq w_h$
              transmit $f$
MG′ (step $t$)
    $O_t \leftarrow$ oblivious schedule at $t$
    $e \leftarrow$ the $\trianglelefteq$-minimal packet from $O_t$
    $h \leftarrow$ the $\trianglelefteq$-minimal of all the heaviest packets from $O_t$
    if $\phi w_e \geq w_h$
        then transmit $e$
        else  transmit $h$
RG (step $t$)
    $O_t \leftarrow$ oblivious schedule at $t$
    $e \leftarrow$ the $\trianglelefteq$-minimal packet from $O_t$
    $h \leftarrow$ the $\trianglelefteq$-minimal of all the heaviest packets from $O_t$
    transmit $e$ with probability $\frac{w_e}{w_h}$ and $h$ with probability $1 - \frac{w_e}{w_h}$
```

Figure 4: The three algorithms

3.2. Analysis Idea

The analysis of Li et al. [12] uses the following idea: in each step, after both MG and Adv transmitted their packets, modify Adv's buffer in such a way that it remains the same as MG's and that this change can only improve Adv's gain, both in this step and in the future. Sometimes Adv's schedule is also modified to achieve this goal, specifically, the packets in it may be reordered, and Adv may sometimes be allowed to transmit two packets in a single step. It is proved that in each such step the ratio of Adv's to MG's gain is at most ϕ. As was already noticed by Li et al. [12], this is essentially a potential function argument. To simplify the analysis, it is assumed (wlog) that Adv transmits its packets in the $\trianglelefteq$ order.

Our analysis follows the outline of the one by Li et al., but we make it more formal. Observe that there may be multiple clairvoyant schedules, and that Adv can transmit $C_t(t)$ at every step t, where C_t is a clairvoyant schedule chosen arbitrarily in step t. As our algorithms MG$'$ and RG determine the oblivious schedule O_t at each step, we assume that every C_t is a clairvoyant schedule conforming with O_t.

There is one exception though. Sometimes, when a reordering in C_t does not hinder Adv's performance (taking future arrivals into account), we "force" Adv to follow the reordered schedule. This is the situation described in Lemma 2.7: when $e \notin C_t$, there is a clairvoyant schedule C^*_t such that $h = C^*_t(t)$. In such case we may assume that Adv follows C^*_t rather than C_t, i.e., that it transmits h at t. Indeed, we make that assumption whenever our algorithm (either MG$'$ or RG) transmits h at such step: then Adv and MG$'$ (RG) transmit the same packet, which greatly simplifies the analysis.

Our analysis of MG$'$ is essentially the same as the original analysis of MG by Li et al. [12], but lacks one case which is superfluous due to our modification. As our algorithm MG$'$ always transmits either e or h, and the packet j that Adv transmits always satisfies $j \trianglelefteq h$ by definition of the clairvoyant schedule conforming with O_t, the case which MG transmits f such that $e \triangleleft f \triangleleft j$ does not occur to MG$'$. The same observation applies to RG, whose analysis also follows the ideas of Li et al.

3.3. Analysis of the Deterministic Algorithm

We analyze this algorithm as mentioned before, i.e., assuming (wlog) that at every step t Adv transmits $C_t(t)$, where C_t is a clairvoyant schedule conforming with O_t.

Theorem 3.1. MG$'$ *is ϕ-competitive on sequences with agreeable deadlines.*

Proof. Note that whenever MG' and Adv transmit the same packet, clearly their gains are the same, as are their buffers right after such step. In particular this happens when $e = h$ as then MG$'$ transmits $e = h$ and Adv does the same: in such case h is both the heaviest packet and the $\trianglelefteq$-minimal non-dominated packet, so $h = C_t(t)$ by definition of the clairvoyant schedule conforming with O_t.

In what follows we inspect the three remaining cases.

$\boldsymbol{\phi w_e \geq w_h}$**:** **MG$'$ transmits $\boldsymbol{e}$.** Adv transmits $\boldsymbol{j \neq e}$.. To make the buffers of MG$'$ and Adv identical right after this step, we replace e in Adv's buffer by j. This is advantageous for Adv as $d_j \geq d_e$ and $w_j \geq w_e$ follows from $e \trianglelefteq j$ and the definition of a clairvoyant schedule conforming with O_t. As $\phi w_e \geq w_h$, the ratio of gains is

$$\frac{w_j}{w_e} \leq \frac{w_h}{w_e} \leq \phi \ .$$

$\boldsymbol{\phi w_e < w_h}$**:** MG′ transmits $\boldsymbol{h}$**.** Adv transmits $\boldsymbol{e}$**.**. Note that Adv's clairvoyant schedule from this step contains h by Fact 2.6. We let Adv transmit both e and h in this step and keep e in its buffer, making it identical to the buffer of MG′. Keeping e, as well as transmitting two packets at a time is clearly advantageous for Adv. As $\phi w_e < w_h$, the ratio of gains is

$$\frac{w_e + w_h}{w_h} \le \frac{1}{\phi} + 1 = \phi \ .$$

$\boldsymbol{\phi w_e < w_h}$**:** MG′ transmits $\boldsymbol{h}$**.** Adv transmits $\boldsymbol{j \neq e}$**.**. Note that $j \trianglelefteq h$: by definition of the clairvoyant schedule conforming with O_t, for every $i \in O_t$ such that $i \triangleleft j$, $w_i < w_j$ holds.

There are two cases: either $j = h$, or $w_j < w_h$ and $d_j < d_h$. In the former one both players do the same and end up with identical buffers. Thus we focus on the latter case. Fact 2.6 implies that $h \in C_t$. By Lemma 2.7, C_t remains feasible when h is moved to its very beginning. Hence we assume that Adv transmits h in the current step. As this is the packet that MG′ sends, the gains of Adv and MG′ are the same and no changes need be made to Adv's buffer. ∎

3.4. Analysis of the Randomized Algorithm

We analyze this algorithm as mentioned before, i.e., assuming (wlog) that at every step t Adv transmits $C_t(t)$, where C_t is a clairvoyant schedule conforming with O_t.

Theorem 3.2. RG *is* $\frac{4}{3}$*-competitive against oblivious adversary on sequences with agreeable deadlines.*

Proof. Observe that if $e = h$, then RG transmits $e = h$ and Adv does the same: as in such case h is both the heaviest packet and the $\trianglelefteq$-minimal non-dominated packet, $h = C_t(t)$ by definition of the clairvoyant schedule conforming with O_t. In such case the gains of RG and Adv are clearly the same, as are their buffers right after step t. Thus we assume $e \neq h$ from now on.

Let us first bound the algorithm's expected gain in one step. It equals

$$\begin{aligned} \mathcal{G}_{\mathrm{RG}} &= \frac{w_e}{w_h} \cdot w_e + \left(1 - \frac{w_e}{w_h}\right) \cdot w_h \\ &= \frac{1}{w_h}\left(w_e^2 - w_e w_h + w_h^2\right) \\ &= \frac{1}{w_h}\left(\left(w_e - \frac{w_h}{2}\right)^2 + \frac{3}{4} w_h^2\right) \\ &\geq \frac{3}{4} w_h \ . \end{aligned} \tag{3.1}$$

Now we describe the changes to Adv's scheduling policy and buffer in the given step. These make Adv's RG's buffers identical, and, furthermore, make the expected gain of the adversary equal exactly w_h. This, together with (3.1) yields the desired bound. To this end we consider cases depending on Adv's choice.

(1) ADV transmits e. Note that ADV's clairvoyant schedule from this step contains h by Fact 2.6.

If RG transmits e, which it does with probability $\frac{w_e}{w_h}$, both players gain w_e and no changes are required.

Otherwise RG transmits h, and we let ADV transmit both e and h in this step and keep e in its buffer, making it identical to RG's buffer. Keeping e, as well as transmitting two packets at a time is clearly advantageous for ADV.

Thus in this case the adversary's expected gain is

$$\mathcal{G}_{\text{ADV}} = \frac{w_e}{w_h} \cdot w_e + \left(1 - \frac{w_e}{w_h}\right)(w_e + w_h) = w_e + (w_h - w_e) = w_h \ .$$

(2) ADV transmits $j \neq e$. Note that $j \trianglelefteq h$: by definition of the clairvoyant schedule conforming with O_t, for every $i \in O_t$ such that $i \triangleleft j$, $w_i < w_j$ holds.

If RG sends e, which it does with probability $\frac{w_e}{w_h}$, we simply replace e in ADV's buffer by j. This is advantageous for ADV as $w_j > w_e$ and $d_j > d_e$ follow from $e \triangleleft j$ and the definition of the clairvoyant schedule conforming with O_t.

Otherwise RG sends h, and we claim that (wlog) ADV does the same. Suppose that $j \neq h$, which implies that $w_j < w_h$ and $d_j < d_h$. Then $h \in C_t$, by Fact 2.6. Thus, by Lemma 2.7, C_t remains feasible when h is moved to its very beginning. Hence we assume that ADV transmits h in the current step. No further changes need be made to ADV's buffer as RG also sends h.

Thus in this case the adversary's expected gain is w_h. ■

4. Conclusion and Open Problems

We have shown that, as long as the adversary is oblivious, the ideas of Li et al. [12] can be applied to randomized algorithms, and devised a $\frac{4}{3}$-competitive algorithm this way. However, the gap between the $\frac{5}{4}$ lower bound and our $\frac{4}{3}$ upper bound remains.

Some parts of our analysis hold even in the adaptive adversary model [8]. On the other hand, other parts do not extend to adaptive adversary model, since in general such adversary's schedule is a random variable depending on the algorithm's random choices. Therefore it is not possible to assume that this "schedule" is ordered by deadlines, let alone perform reordering like the one in proof of Lemma 2.7.

This makes bridging either the $\left[\frac{5}{4}, \frac{4}{3}\right]$ gap in the oblivious adversary model, or the $\left[\frac{4}{3}, \frac{e}{e-1}\right]$ gap in the adaptive adversary model all the more interesting.

Acknowledgements

I would like to thank my brother, Artur Jeż, for numerous comments on the draft of this paper.

References

[1] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994. Also appeared in *Proc. of the 22nd STOC*, 1990.

[2] M. Bieńkowski, M. Chrobak, and Ł. Jeż. Randomized algorithms for Buffer Management with 2-Bounded Delay. In *Proc. of the 6th WAOA*, pages 92–104, 2008.

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

[4] F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall, and T. Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *J. Discrete Algorithms*, 4(2):255–276, 2006.

[5] F. Y. L. Chin and S. P. Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37(3):149–164, 2003.

[6] M. Englert and M. Westermann. Considering suppressed packets improves buffer management in QoS switches. In *Proc. of the 18th SODA*, pages 209–218, 2007.

[7] B. Hajek. On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. In *Conference in Information Sciences and Systems*, pages 434–438, 2001.

[8] Ł. Jeż. Randomised buffer management with bounded delay against adaptive adversary. *CoRR*, abs/0907.2050, 2009.

[9] J. Jeżabek. Resource augmentation for QoS Buffer Management with Agreeable Deadlines. Unpublished, 2009^{+}.

[10] J. Jeżabek. Increasing machine speed in on-line scheduling of weighted unit-length jobs in slotted time. In *Proc. of the 35th SOFSEM*, pages 329–340, 2009.

[11] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. *SIAM J. Comput.*, 33(3):563–583, 2004.

[12] F. Li, J. Sethuraman, and C. Stein. An optimal online algorithm for packet scheduling with agreeable deadlines. In *Proc. of the 16th SODA*, pages 801–802, 2005.

[13] A. Schrijver. *Combinatorial Optimization*, volume A. Springer, 2003.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 501-512
www.stacs-conf.org

COLLAPSIBLE PUSHDOWN GRAPHS OF LEVEL 2 ARE TREE-AUTOMATIC

ALEXANDER KARTZOW [1]

[1] TU Darmstadt, Fachbereich Mathematik, Schlossgartenstr. 7, 64289 Darmstadt, Germany

Abstract. We show that graphs generated by collapsible pushdown systems of level 2 are tree-automatic. Even when we allow ε-contractions and add a reachability predicate (with regular constraints) for pairs of configurations, the structures remain tree-automatic. Hence, their FO theories are decidable, even when expanded by a reachability predicate. As a corollary, we obtain the tree-automaticity of the second level of the Caucal-hierarchy.

1. Introduction

Higher-order pushdown systems were first introduced by Maslov [10, 11] as accepting devices for word languages. Later, Knapik et al. [8] studied them as generators for trees. They obtained an equi-expressivity result for higher-order pushdown systems and for higher-order recursion schemes that satisfy the constraint of *safety*, which is a rather unnatural syntactic condition. Recently, Hague et al. [6] introduced collapsible pushdown systems as extensions of higher-order pushdown systems and proved that these have exactly the same power as higher-order recursion schemes as methods for generating trees.

Both – higher-order and collapsible pushdown systems – also form interesting devices for generating graphs. Carayol and Wöhrle [3] showed that the graphs generated by higher-order pushdown systems[1] of level l coincide with the graphs in the l-th level of the Caucal-hierarchy, a class of graphs introduced by Caucal [4]. Every level of this hierarchy is obtained from the preceding level by applying graph unfoldings and MSO interpretations. Both operations preserve the decidability of the MSO theory whence the Caucal-hierarchy forms a rather large class of graphs with decidable MSO theories. If we use collapsible pushdown systems as generators for graphs we obtain a different situation. Hague et al. showed that even the second level of the hierarchy contains a graph with undecidable MSO theory. But they showed the decidability of the modal μ-calculus theories of all graphs in the hierarchy. This turns graphs generated by collapsible pushdown systems into an interesting class from a model theoretic point of view. There are few natural classes that share these properties. In fact, the author only knows one further example, viz. nested pushdown trees. Alur et al.[1] introduced these graphs for μ-calculus model checking purposes. We

1998 ACM Subject Classification: F.4.1[Theory of Computation]:Mathematical Logic.

Key words and phrases: tree-automatic structures, collapsible pushdown graphs, collapsible pushdown systems, first-order decidability, reachability.

[1] The graph generated by a higher-order pushdown system is the ε-closure of its reachable configurations.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2480

proved in [7] that nested pushdown trees also have decidable first-order theories. We gave an effective model checking algorithm using pumping techniques, but we also proved that nested pushdown trees are tree-automatic structures. Tree-automatic structures were introduced by Blumensath [2]. These structures enjoy decidable first-order theories due to the good closure properties of finite automata on trees.

In this paper, we are going to extend our previous result to the second level of the collapsible pushdown hierarchy. All graphs of the second level are tree-automatic. This subsumes our previous result as nested pushdown trees are first-order interpretable in collapsible pushdown graphs of level two. Furthermore, we show that collapsible pushdown graphs of level 2 are still tree-automatic when expanded by a reachability predicate, i.e., by the binary relation which contains all pairs of configurations such that there is a path from the first to the second configuration. Thus, first-order logic extended by reachability predicates is decidable on level 2 collapsible pushdown graphs.

In the next section, we introduce the necessary notions concerning tree-automaticity and in Section 3 we define collapsible pushdown graphs. We explain the translation of configurations into trees in Section 4. Section 5 is a sketch of the proof that this translation yields tree-automatic representations of collapsible pushdown graphs, even when enriched with certain regular reachability predicates. The last section contains some concluding remarks about questions arising from our result.

2. Preliminaries

We write MSO for monadic second order logic and FO for first-order logic. For words $w_1, w_2 \in \Sigma^*$, we write $w_1 \sqcap w_2$ for the greatest common prefix of w_1 and w_2. A Σ-*labelled tree* is a function $T : D \to \Sigma$ for a finite $D \subseteq \{0,1\}^*$ which is closed under prefixes.
For $d \in D$ we denote by T_d the *subtree rooted at d*.

Sometimes it is useful to define trees inductively by describing their left and right subtrees. For this purpose we fix the following notation. Let $\hat{T}_0$ and $\hat{T}_1$ be Σ-labelled trees and $\sigma \in \Sigma$. Then we write $T := \sigma(\hat{T}_0, \hat{T}_1)$ for the Σ-labelled tree T with the following three properties

1. $T(\varepsilon) = \sigma$, 2. $T_0 = \hat{T}_0$, and 3. $T_1 = \hat{T}_1$.

In the rest of this section, we briefly present the notion of a tree-automatic structure as introduced by Blumensath [2].

The *convolution* of two Σ-labelled trees T and T' is given by a function

$$T \otimes T' : \mathrm{dom}(T) \cup \mathrm{dom}(T') \to (\Sigma \cup \{\Box\})^2$$

where $\Box$ is a new symbol for padding and

$$(T \otimes T')(d) := \begin{cases} (T(d), T'(d)) & \text{if } d \in \mathrm{dom}(T) \cap \mathrm{dom}(T') \\ (T(d), \Box) & \text{if } d \in \mathrm{dom}(T) \setminus \mathrm{dom}(T') \\ (\Box, T'(d)) & \text{if } d \in \mathrm{dom}(T') \setminus \mathrm{dom}(T) \end{cases}$$

By "tree-automata" we mean a nondeterministic finite automaton that labels a finite tree top-down.

Definition 2.1. A structure $\mathfrak{B} = (B, E_1, E_2, \ldots, E_n)$ with domain B and binary relations E_i is *tree-automatic* if there are tree-automata $A_B, A_{E_1}, A_{E_2}, \ldots, A_{E_n}$ and a bijection

$f : L \to B$ for L the language accepted by A_B such that the following hold. For $T, T' \in L$, the automaton A_{E_i} accepts $T \otimes T'$ if and only if $\big(f(T), f(T')\big) \in E_i$.

Tree-automatic structures form a nice class because automata theoretic techniques may be used to decide first-order formulas on these structures:

Lemma 2.2 ([2])**.** *If B is tree-automatic, then its first-order theory is decidable.*

We will use the classical result that regular sets of trees are MSO definable.

Theorem 2.3 ([12], [5])**.** *For a set $\mathbb{T}$ of finite Σ-labelled trees, there is a tree automaton recognising $\mathbb{T}$ if and only if $\mathbb{T}$ is* MSO *definable.*

3. Definition of Collapsible Pushdown Graphs (CPG)

In this section we define our notation of collapsible pushdown systems. For a more comprehensive introduction, we refer the reader to [6].

3.1. Collapsible Pushdown Stacks

First, we provide some terminology concerning stacks of (collapsible) higher-order pushdown systems. We write Σ^{*2} for $(\Sigma^*)^*$ and Σ^{+2} for $(\Sigma^+)^+$. We call an $s \in \Sigma^{*2}$ a 2-word.

Let us fix a 2-word $s \in \Sigma^{*2}$ which consists of an ordered list $w_1, w_2, \dots, w_m \in \Sigma^*$. We separate the words of this list by colons writing $s = w_1 : w_2 : \dots : w_m$. By $|s|$ we denote the number of words s consists of, i.e., $|s| = m$.

For another word $s' = w'_1 : w'_2 : \dots : w'_n \in \Sigma^{*2}$, we write $s : s'$ for the concatenation $w_1 : w_2 : \dots : w_m : w'_1 : w'_2 : \dots : w'_n$.

If $w \in \Sigma^*$, we write $[w]$ for the 2-word that consists of a list of one word which is w.

A level 2 collapsible pushdown stack is a special element of $(\Sigma \times \{1,2\} \times \mathbb{N})^{+2}$ that is generated by certain stack operations from an initial stack which we introduce in the following definitions. The natural numbers following the stack symbol represent the so-called *collapse pointer*: every element in a collapsible pushdown stack has a pointer to some substack and applying the collapse operation returns the substack to which the topmost symbol of the stack points. Here, the first number denotes the *collapse level.* If it is 1 the collapse pointer always points to the symbol below the topmost symbol and the collapse operations just removes the topmost symbol. The more interesting case is when the collapse level of the topmost symbol of the stack s is 2. Then the stack obtained by the collapse contains the first n words of s where n is the second number in the topmost element of s.

The initial level 1 stack is $\bot_1 := (\bot, 1, 0)$ and the initial level 2 stack is $\bot_2 := [\bot_1]$.

For $k \in \{1, 2\}$ and for a 2-word $s = w_1 : w_2 : \dots : w_n \in (\Sigma \times \{1,2\} \times \mathbb{N})^{+2}$ such that $w_n = a_1 a_2 \dots a_m$ with $a_i \in \Sigma \times \{1,2\} \times \mathbb{N}$ for all $1 \le i \le m$:

- we define the *topmost $(k-1)$-word of s* as $\mathrm{top}_k(s) := \begin{cases} w_n & \text{if } k = 2 \\ a_m & \text{if } k = 1 \end{cases}$
- for $\mathrm{top}_1(s) = (\sigma, i, j) \in \Sigma \times \{1,2\} \times \mathbb{N}$, we define the *topmost symbol* $\mathrm{Sym}(s) := \sigma$, the *collapse-level of the topmost element* $\mathrm{CLvl}(s) := i$, and the *collapse-link of the topmost element* $\mathrm{CLnk}(s) := j$.

For s, w_n and k as before, $\sigma \in \Sigma \backslash \{\bot\}$, and $w'_n := a_1 \dots a_{m-1}$, we define the stack operations

$$\mathrm{pop}_k(s) := \begin{cases} w_1 : w_2 : \dots : w_{n-1} & \text{if } k = 2, n \geq 2 \\ w_1 : w_2 : \dots : w_{n-1} : w'_n & \text{if } k = 1, m \geq 2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathrm{clone}_2(s) := w_1 : w_2 : \dots : w_{n-1} : w_n : w_n$$

$$\mathrm{push}_{\sigma,k}(s) := \begin{cases} w_1 : w_2 : \dots : w_n(\sigma, 2, n-1) & \text{if k=2} \\ w_1 : w_2 : \dots : w_n(\sigma, 1, m) & \text{if k=1} \end{cases}$$

$$\mathrm{collapse}(s) := \begin{cases} w_1 : w_2 : \dots : w_r & \text{if } \mathrm{CLvl}(s) = 2, \mathrm{CLnk}(s) = r > 0 \\ \mathrm{pop}_1(s) & \text{if } \mathrm{CLvl}(s) = 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

The *set of level* 2*-operations* is $\mathrm{OP} := \big\{\mathrm{push}_{\sigma,1}, \mathrm{push}_{\sigma,2}, \mathrm{clone}_2, \mathrm{pop}_1, \mathrm{pop}_2, \mathrm{collapse}\big\}$. The *set of level* 2 *stacks*, $\mathrm{Stck}(\Sigma)$, is the smallest set that contains $\bot_2$ and is closed under all operations from OP.

Note that collapse- and pop_k-operations are only allowed if the resulting stack is in $(\Sigma^+)^+$. This avoids the special treatment of empty words or stacks. Furthermore, a collapse on level 2 summarises a non-empty sequence of pop_2-operations. For example, starting from $\bot_2$, we can apply a clone_2, a $\mathrm{push}_{\sigma,2}$, a clone_2, and finally a collapse. This sequence first creates a level 2 stack that contains 3 words and then performs the collapse and ends in the initial stack again. This example shows that clone_2-operations are responsible for the fact that collapse-operations on level 2 may remove more than one word from the stack.

For $s, s' \in \mathrm{Stck}(\Sigma)$, we call s' a substack of s if there are $n_1, n_2 \in \mathbb{N}$ such that $s' = {\mathrm{pop}_1}^{n_1}({\mathrm{pop}_2}^{n_2}(s))$. We write $s' \leq s$ if s' is a substack of s.

3.2. Collapsible Pushdown Systems and Collapsible Pushdown Graphs

Now we introduce collapsible pushdown systems and graphs (of level 2) which are analogues of pushdown systems and pushdown graphs using collapsible pushdown stacks instead of ordinary stacks.

Definition 3.1. A *collapsible pushdown system* of level 2 (CPS) is a tuple $S = (\Sigma, Q, \Delta, q_0)$ where Σ is a finite stack alphabet with $\bot \in \Sigma$, Q a finite set of states, $q_0 \in Q$ the initial state, and $\Delta \subseteq Q \times \Sigma \times Q \times \mathrm{OP}$ the transition relation.

For $q \in Q$ and $s \in \mathrm{Stck}(\Sigma)$ the pair (q, s) is called a *configuration.* We define labelled transitions on pairs of configurations by setting $(q_1, s) \vdash^{(q_2, op)} (q_2, t)$ if there is a $(q_1, \sigma, q_2, op) \in \Delta$ such that $\mathrm{Sym}(s) = \sigma$ and $op(s) = t$. The union of the labelled transition relations is denoted as $\vdash := \bigcup_{l \in Q \times \mathrm{OP}} \vdash^l$. We set $C(S)$ to be the set of all configurations that are reachable from $(q_0, \bot_2)$ via $\vdash$-paths. We call $C(S)$ the set of *reachable* or *valid* configurations. The *collapsible pushdown graph (*CPG*) generated by S* is

$$\mathrm{CPG}(S) := \Big(C(S), (C(S)^2 \cap \vdash^{\ell})_{\ell \in Q \times \mathrm{OP}}\Big)$$

Example 3.2. The following example of a collapsible pushdown graph of level 2 is taken from [6]. Let $Q := \{0, 1, 2\}, \Sigma := \{\bot, a\}$, and Δ given by $(0, *, 1, \mathrm{clone}_2)$, $(1, *, 0, \mathrm{push}_{a,2})$, $(1, *, 2, \mathrm{push}_{a,2})$, $(2, a, 2, \mathrm{pop}_1)$, and $(2, a, 0, \mathrm{collapse})$, where $*$ denotes any letter in Σ. In our picture (see Figure 1), the labels are abbreviated as follows: $\mathrm{cl} := (1, \mathrm{clone}_2)$, $a := (0, \mathrm{push}_{a,2})$,

$a' := (2, \text{push}_{a,2})$, $p := (2, \text{pop}_1)$, and $\text{co} := (0, \text{collapse})$.

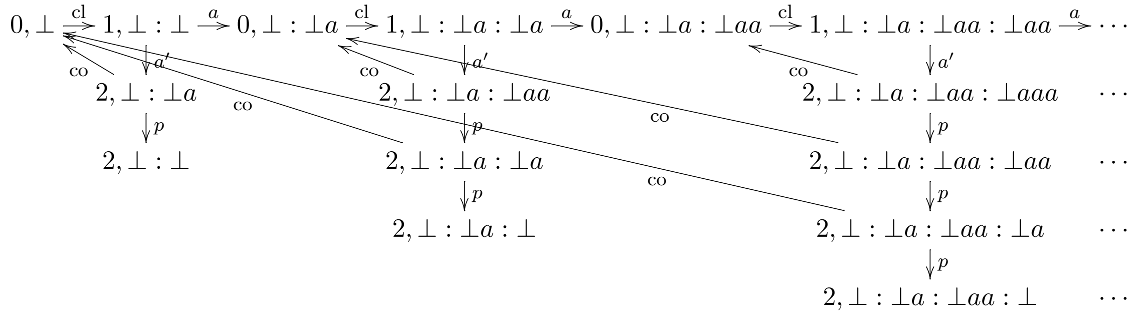

$0, \bot$
cl
$1, \bot : \bot$
a
$0, \bot : \bot a$
cl
$1, \bot : \bot a : \bot a$
a
$0, \bot : \bot a : \bot aa$
cl
$1, \bot : \bot a : \bot aa : \bot aa$
a
$\cdots$
co
a'
co
a'
co
a'
$2, \bot : \bot a$
co
$2, \bot : \bot a : \bot aa$
co
$2, \bot : \bot a : \bot aa : \bot aaa$
$\cdots$
p
p
p
$2, \bot : \bot$
$2, \bot : \bot a : \bot a$
co
$2, \bot : \bot a : \bot aa : \bot aa$
$\cdots$
p
p
$2, \bot : \bot a : \bot$
$2, \bot : \bot a : \bot aa : \bot a$
$\cdots$
p
$2, \bot : \bot a : \bot aa : \bot$
$\cdots$

Figure 1: Example of a collapsible pushdown graph

Remark 3.3. Hague et al. [6] showed that modal μ-calculus model checking on level n CPG is n-EXPTIME complete. Note that there is an MSO interpretation which turns the graph of the previous example into a grid-like structure. Hence its MSO theory is undecidable.

The next definition introduces runs of collapsible pushdown systems.

Definition 3.4. Let S be a CPS. A run r of S of length n is a function

$$r : \{0, 1, 2, \ldots, n\} \to Q \times (\Sigma \times \{1, 2\} \times \mathbb{N})^{*2} \text{ such that } r(0) \vdash r(1) \vdash \cdots \vdash r(n).$$

We write $\ln(r) := n$ and call r a run from $r(0)$ to $r(n)$. We say r visits a stack s at i if $r(i) = (q, s)$.

For runs r, r' of length n and m, respectively, with $r(n) = r'(0)$, we define the composition $r \circ r'$ of r and r' in the obvious manner.

Remark 3.5. Note that we do not require runs to start in the initial configuration.

4. Encoding of Collapsible Pushdown Graphs in Trees

In this section we prove that CPG are tree-automatic. For this purpose we have to encode stacks in trees. The idea is to divide a stack into *blocks* and to encode different blocks in different subtrees. The crucial observation is that every stack is a list of words that share the same first letter. A block is a maximal list of words in the stack that share the same two first letters[2]. If we remove the first letter of every word of such a block, the resulting 2-word decomposes again as a list of blocks. Thus, we can inductively carry on to decompose parts of a stack into blocks and code every block in a different subtree. The roots of these subtrees are labelled with the first letter of the corresponding block. This results in a tree in which every initial left-closed path represents one word of the stack. By left-closed, we mean that the last element of the path has no left successor.

It turns out that – via this encoding – each stack operation corresponds to a simple MSO-definable tree-operation. The main difficulty is to provide a tree-automaton that checks whether there is a run to the configuration represented by some tree. This problem is addressed in Section 5.

[2] see Figure 2 for an example of blocks and Definition 4.1 for their formal definition

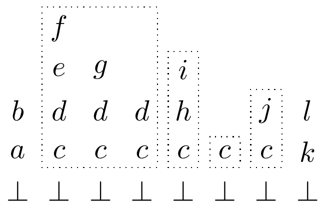


Figure 2: Example of blocks in a stack. These form a c-blockline.

As already mentioned, the encoding works by dividing stacks into blocks. The following definition makes our notion of blocks precise. For $w \in \Sigma^*$ and $s = w_1 : w_2 : \ldots : w_n \in \Sigma^{*2}$, we write $s' := w \setminus s$ for $s' = [ww_1] : [ww_2] : \ldots : [ww_n]$.

Definition 4.1 (σ-block(line))**.** For $\sigma \in \Sigma$, we call $b \in \Sigma^{*2}$ a *σ-block* if $b = [\sigma]$ or $b = \sigma\tau \setminus s'$ for some $\tau \in \Sigma$ and $s' \in \Sigma^{*2}$. See Figure 2 for examples of blocks. If $b_1, b_2, \ldots, b_n$ are σ-blocks, then we call $b_1 : b_2 : \ldots : b_n$ a *σ-blockline.*

Note that every stack in $\mathrm{Stck}(\Sigma)$ forms a $(\bot, 1, 0)$-blockline. Furthermore, every blockline l decomposes uniquely as $l = b_1 : b_2 : \ldots : b_n$ of maximal blocks b_i in l. Another crucial observation is that a σ-block $b \in \Sigma^{*2} \setminus \Sigma$ decomposes as $b = \sigma \setminus l$ for some blockline l and we say l is the induced blockline of b. For $b \in \Sigma$ the induced blockline of $[b]$ is just the empty 2-word.

Now we encode a (σ, n, m)-blockline l in a tree by labelling the root with (σ, n), by encoding the blockline induced by the first block of l in the left subtree, and by encoding the rest of the blockline in the right subtree. In order to avoid repetitions, we do not repeat the symbol (σ, n) in the right subtree, but replace it by the default letter ε.

Definition 4.2. Let $s = w_1 : w_2 : \ldots : w_n \in (\Sigma \times \{1,2\} \times \mathbb{N})^{+2}$ be a (σ, l, k)-blockline. Let w'_i be words such that $s = (\sigma, l, k) \setminus [w'_1 : w'_2 : \ldots : w'_n]$ and set $s' := w'_1 : w'_2 : \ldots : w'_n$. As an abbreviation we write ${}_hs_i := w_h : w_{h+1} : \ldots : w_i$. Furthermore, let $w_1 : w_2 : \ldots : w_j$ be a maximal block of s. Note that $j > 1$ implies $w_{j'} = (\sigma, l, k)(\sigma', l', k')w''_{j'}$ for all $j' \leq j$, some fixed $(\sigma', l', k') \in \Sigma \times \{1,2\} \times \mathbb{N}$, and appropriate $w''_{j'} \in \Sigma^*$. For $\rho \in \big(\Sigma \times \{1,2\}\big) \cup \{\varepsilon\}$, we define recursively the $\big(\Sigma \times \{1,2\}\big) \cup \{\varepsilon\}$-labelled tree $\mathrm{Enc}(s, \rho)$ via

$$\mathrm{Enc}(s, \rho) := \begin{cases} \rho & \text{if } |w_1| = 1, n = 1 \\ \rho(\emptyset, \mathrm{Enc}({}_2s_n, \varepsilon)) & \text{if } |w_1| = 1, n > 1 \\ \rho(\mathrm{Enc}({}_1s'_n, (\sigma', l')), \emptyset) & \text{if } j = n, |w_1| > 1 \\ \rho(\mathrm{Enc}({}_1s'_j, (\sigma', l')), \mathrm{Enc}({}_{j+1}s_n, \varepsilon)) & \text{otherwise.} \end{cases}$$

$\mathrm{Enc}(s) := \mathrm{Enc}(s, (\bot, 1))$ is called the (tree-)encoding of the stack $s \in \mathrm{Stck}(\Sigma)$.

Figure 3 shows a configuration and its encoding.

Remark 4.3. In this encoding, the first block of a (σ, l, k)-blockline is encoded in a subtree whose root d is labelled (σ, l). We can restore k from the position of d in the tree $\mathrm{Enc}(s)$ as follows. If $l = 1$ then $k = |d|_0$, i.e., the number of occurrences of 0 in d. This is due to the fact that level 1 links always point to the preceding letter and that we always introduce a left-successor tree in order to encode letters that are higher in the stack.

The case $l = 2$ needs some closer inspection. Assume that some $d \in T := \mathrm{Enc}(s)$ is labelled $(\sigma, 2)$. Then it encodes a letter $(\sigma, 2, k)$ and this is not a cloned element.

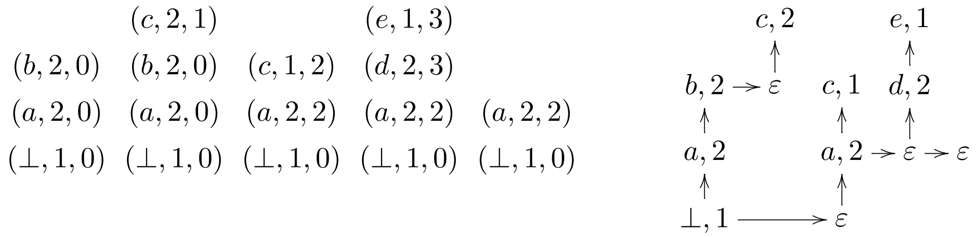


Figure 3: A stack s and its encoding Enc(s): right arrows lead to 1-successors (right successors), upward arrows lead to 0-successors (left successors).

Thus, k equals the numbers of words to the left of this letter $(\sigma, 2, k)$. We claim that $k = \left|\left\{e \in T \cap \{0,1\}^*1 : e \leq_{lex} d\right\}\right|$. The existence of a pair $e, e1 \in T$ corresponds to the fact that there is some blockline consisting of blocks $b_1 : b_2 : \ldots : b_n$ with $n \geq 2$ such that b_1 is encoded in $T_e \setminus T_{e1}$ and $b_2 : \ldots : b_n$ is encoded in T_{e1}. By induction, one easily sees that for each such pair $e, e1 \in T$ all the letters that are in words left of the letter encoded by $e1$ are encoded in lexicographically smaller elements. Furthermore, the size of $((0^*)1)^* \cap T$ corresponds to the number of words in s since the introduction of a 1-successor corresponds to the separation of the first block of some blockline from the other blocks. Each of these separation can also be seen as the separation of the last word of the first block from the first word of the second block of this blockline. Note that we separate two words that are next to each other in exactly one blockline. Putting these facts together our claim is proved.

Another view on this correspondence is the bijection $f : \{1, 2, \ldots, |s|\} \to R$ where $R := ((0^*)1)^* \cap \mathrm{dom}(T)$ and i is mapped to the i-th element of R in lexicographic order. $f(i)$ is exactly the position where the $(i-1)$-st word is separated from the i-th one for all $i \geq 2$. In order to state the properties of f, we need some more notation. We write π for the canonical projection $\pi : (\Sigma \times \{1,2\} \times \mathbb{N})^* \to (\Sigma \times \{1,2\})^*$ and w_i for the i-th word of s. Furthermore, let w'_i be a word such that, $w_i = (w_i \sqcap w_{i-1}) \circ w'_i$ (here we set $w_0 := \varepsilon$). Then the word along the path[3] from the root to $f(i)$ is exactly $\pi(w_i \sqcap w_{i-1})$ for all $2 \leq i \leq |s|$ and the path from $f(j)$ to $f(j) \circ 0^m$ for maximal $m \in \mathbb{N}$ is $\pi(w'_j)$ for all $1 \leq j \leq |s|$.

In order to encode a configuration $c := (q, s)$, we add q as a new root of the tree and attach the encoding of s as the left subtree, i.e., $\mathrm{Enc}(c) := q(\mathrm{Enc}(s), \emptyset)$.

The image of this encoding function contains only trees of a very specific type. We call this class $\mathbb{T}_{\mathrm{Enc}}$. In the next definition we state the characterising properties of $\mathbb{T}_{\mathrm{Enc}}$. This class is MSO definable, whence automata-recognisable.

Definition 4.4. Let $\mathbb{T}_{\mathrm{Enc}}$ be the class of all trees T that satisfy the following conditions.

(1) The root of T is labelled by some element of Q $(T(\varepsilon) \in Q)$.
(2) Every element of the form $\{0,1\}^*0$ is labelled by some $(\sigma, l) \in \Sigma \times \{1,2\}$; especially, $T(0) = (\bot, 1)$ and there are no other occurrences of $(\bot, 1)$ or $(\bot, 2)$.
(3) Every element of the form $\{0,1\}^*1$ is labelled by ε.
(4) $1 \notin \mathrm{dom}(T)$, $0 \in \mathrm{dom}(T)$.
(5) For all $t \in T$, if $T(t0) = (\sigma, 1)$ then $T(t10) \neq (\sigma, 1)$.

[3] By the word along a path from one node to another we mean the word consisting of the non ε-labels along this path.

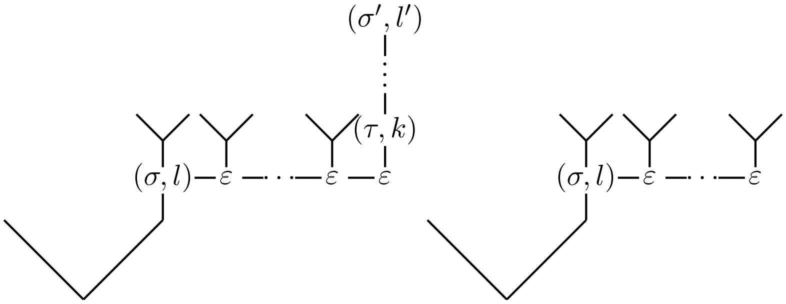

(σ', l')
(τ, k)
(σ, l)—ε —· ·—ε —ε
(σ, l)—ε —· ·—ε

Figure 4: pop_2-operation

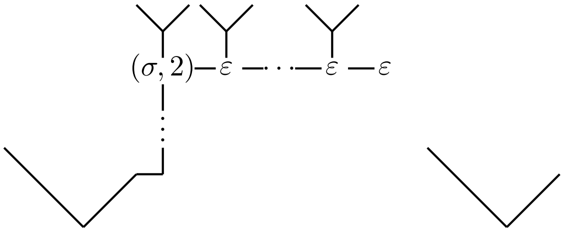

$(\sigma, 2)$—ε —· ·—ε —ε

Figure 5: collapse-operation of level 2.

Remark 4.5. Note that (5) holds as $T(t0) = T(t10) = (\sigma, 1)$ would imply that the subtree rooted at t encodes a blockline l such that the first block of l induces a $(\sigma, 1, n)$-blockline and the second one induces a $(\sigma, 1, m)$-blockline. But as level 1 links always point to the preceding letter, n and m are equal to the length of the prefix of l in the stack plus 1, i.e., if T encodes a stack s then $s = s_1 : [w \setminus l] : s_2$ and $n = m = |w| + 1$. This would contradict the maximality of the blocks in the encoding.

Remark 4.6. $\text{Enc} : Q \times \text{Stck}(\Sigma) \to \mathbb{T}_{\text{Enc}}$ is a bijection and we denote its inverse by Dec.

Our encoding turns the transitions of a CPG into regular tree-operations. The tree-operations corresponding to pop_2 and collapse can be seen in Figures 4 and 5. For the pop_2, note that if v_1 is the 0-successor of v_0 then v_0 and v_1 encode symbols in the same word of the encoded stack. As a pop_2 removes the rightmost word, we have to remove all the nodes encoding information about this word. As the rightmost leaf corresponds to the topmost symbol of the stack, we have to remove this leaf and all its 0-ancestors.

For the collapse (on level 2), we note that each ε represents a cloned element. The collapse induced by such an element produces the same stack as a pop_2 of its original version. The original symbol of the rightmost leaf is its first ancestor not labelled by ε.

Note that the operations corresponding to pop_2 and collapse are clearly MSO definable. All other transitions in CPG correspond to MSO definable tree-operations, too. Due to space restrictions we skip the details.

Lemma 4.7. *Let C be the set of encodings of configurations of a* CPS *S. Then there are automata $A_{(q,\text{op})}$ for all $q \in Q$ and all* $\text{op} \in \text{OP}$ *such that for all $c_1, c_2 \in C$*

$$A_{(q,\text{op})} \textit{ accepts } \text{Enc}(c_1) \otimes \text{Enc}(c_2) \quad \textit{iff} \quad c_1 \vdash^{(q,\text{op})} c_2 \ .$$

5. Recognising Reachable Configurations

We show that Enc maps the reachable configurations of a given CPS to a regular set. For this purpose we introduce milestones of a stack s. It turns out that these are exactly those substacks of s that every run to s has to visit. Furthermore, the milestones of s are represented by the nodes of $\mathrm{Enc}(s)$: with every $d \in \mathrm{Enc}(s)$, we can associate a subtree of s which encodes a milestone. Furthermore, the substack relation on the milestones corresponds exactly to the lexicographical order $\leq_{lex}$ of the elements of $\mathrm{Enc}(s)$. For every $d \in \mathrm{Enc}(s)$ we can guess the state in which the corresponding milestone is visited for the last time by some run to s and we can check the correctness of this guess using MSO or, equivalently, tree-automata.

We prove that we can check the correctness of such a guess by introducing a special type of run, called *loop*, which is basically a run that starts and ends with the same stack. A run from one milestone to the next will mainly consist of loops combined with a finite number of stack operations.

5.1. Milestones

Definition 5.1 (Milestone)**.** A substack s' of $s = w_1 : w_2 : \ldots : w_n$ is a *milestone* if $s' = w_1 : w_2 : \ldots : w_i : w'$ such that $0 \leq i < n$ and $w_i \sqcap w_{i+1} \leq w' \leq w_{i+1}$. We denote by $\mathrm{MS}(s)$ the set of milestones of s.

Note that the substack relation $\leq$ linearly orders $\mathrm{MS}(s)$.

Lemma 5.2. *If s, t, m are stacks with $m \in \mathrm{MS}(t)$ but $m \not\leq s$, then every run from s to t visits m. Thus, for every run r from the initial configuration to s, the function*

$$f : \mathrm{MS}(s) \to dom(r), \qquad s' \mapsto \max\{i \in dom(r) : r(i) = (q, s') \text{ for some } q \in Q\}$$

is an order embedding with respect to substack relation on the milestones and the natural order of $dom(r)$.

In order to state the close correspondence between milestones of a stack s and the elements of $\mathrm{Enc}(s)$, we need the following definition.

Definition 5.3. Let $T \in \mathbb{T}_{\mathrm{Enc}}$ be a tree and $d \in T \setminus \{\varepsilon\}$. Then the *left and downward closed tree induced by* d is $LT(d, T) := T{\restriction}_D$ where $D := \{d' \in T : d' \leq_{lex} d\} \setminus \{\varepsilon\}$. Then we denote by $\mathrm{LStck}(d, T) := \mathrm{Dec}(LT(d, T))$ the *left stack induced by d.*

Remark 5.4. $\mathrm{LStck}(d, s)$ is a substack of s for all $d \in \mathrm{dom}(\mathrm{Enc}(s))$. This observation follows from Remark 4.3 combined with the fact that the left stack is induced by a lexicographically downward closed subset. In fact, $\mathrm{LStck}(d, s)$ is a milestone of s.

Lemma 5.5. *The map given by $g : d \mapsto \mathrm{LStck}(d, \mathrm{Enc}(s))$ is an order isomorphism between $(dom(\mathrm{Enc}(q, s)) \setminus \{\varepsilon\}, \leq_{lex})$ and $(\mathrm{MS}(s), \leq)$.*

Lemmas 5.5 and 5.2 imply that every run r decomposes as $r = r_1 \circ r_2 \circ \ldots \circ r_n$ where r_i is a run from the i-th milestone of $r(\ln(r))$ to the $(i+1)$-st milestone.

In order to describe the structure of the r_i, we have to introduce the notion of a loop. Informally speaking, a loop is a run r that starts and ends with the same stack s and which does not look too much into s.

Definition 5.6. Let r be a run of length n with $r(i) = (q_i, s_i)$ for all $0 \leq i \leq n$.

- r is called a *simple high loop* if $s_0 = s_n$ and if $s_0 < s_i$ for all $0 < i < n$.
- r is called a *simple low loop* of s if $s_0 = s_n = s$, between 0 and n the stack s is never visited, $s_1 = \mathrm{pop}_1(s)$, $\mathrm{CLvl}(s) = 1$, $|s_i| \geq |s|$ for all $0 \leq i \leq n$, and $r{\restriction}_{[2,n-1]}$ is the composition of simple low loops and simple high loops of $\mathrm{pop}_1(s)$.
- r is called *loop* if it is a finite composition of low loops and high loops.

Lemma 5.7. *Let s be some stack, m_1, m_2 milestones of s, and r a run from m_1 to m_2 that never visits any other milestone of s. Then either $r = l_1 \circ p \circ l_2$ or $r = l_0 \circ c \circ l_1 \circ p_1 \circ l_2 \circ p_2 \circ l_3 \circ \ldots \circ p_n \circ l_{n+1}$ where each l_i is a loop, and all p_i, p, and c are runs of length* 1*, p performs one* $\mathrm{push}_{\sigma,k}$*, c performs one* clone_2*, and the p_i perform one* pop_1 *each.*

This lemma motivates why we only define low loops for stacks s with $\mathrm{CLvl}(s) = 1$. Whenever the topmost symbol of a milestone m is not a cloned element, then $\mathrm{pop}_1(m)$ is another milestone. Hence, the l_i can only contain low loops if they start at a stack with cloned topmost symbol. But any stack s with cloned topmost symbol and $\mathrm{CLvl}(s) = 2$ cannot be restored from $\mathrm{pop}_1(s)$ without passing $\mathrm{pop}_2(s)$ since a $\mathrm{push}_{\sigma,2}$-operation would create the wrong link-level.

From Lemma 5.7 we can derive that deciding whether there is a run from one milestone to the next is possible if we know the pairs of initial and final states of loops of certain stacks s. Hence we are interested in the sets $\mathrm{Loops}(s) \subseteq Q \times Q$ with $(q_1, q_2) \in \mathrm{Loops}(s)$ if and only if there is a loop from (q_1, s) to (q_2, s). The crucial observation is that $\mathrm{Loops}(s)$ may be calculated by a finite automaton reading $\mathrm{top}_2(s)$.

Lemma 5.8. *For every* CPS *there exists a finite automaton A that calculates*[4] *on input $w \in (\Sigma \times \{1,2\})^*$ the set* $\mathrm{Loops}(s)$ *for all stacks s such that $w = \pi(\mathrm{top}_2(s))$. Here, $\pi : (\Sigma \times \{1,2\} \times \mathbb{N})^* \to (\Sigma \times \{1,2\})^*$ is the projection onto the symbols and collapse-levels.*

5.2. Detection of Reachable Configurations

We have already seen that every run to a valid configuration (q, s) passes all the milestones of s. Now, we use the last state in which a run r to (q, s) visits each milestone as a certificate for the reachability of (q, s). To be precise, *a certificate for the reachability of* (q, s) is a map $f : \mathrm{dom}\big(\mathrm{Enc}(q, s)\big) \setminus \{\varepsilon\} \to Q$ such that there is some run r from $\bot_2$ to (q, s) and $f(d) = q$ if and only if $r(i) = \big(q, \mathrm{LStck}(d)\big)$ for i the maximal position in r where $\mathrm{LStck}(d)$ is visited.

Lemma 5.9. *For every* CPG *G, there is a tree-automaton that checks for each map*

$$f : dom(\mathrm{Enc}(q, s)) \setminus \{\varepsilon\} \to Q$$

whether f is a certificate of the reachability of (q, s), i.e., whether f is induced by some run r from the initial configuration to (q, s).

The proof of the lemma uses Lemma 5.8 and the fact that the path from the root to some $d \in \mathrm{Enc}(s)$ encodes the topmost word of $\mathrm{LStck}(d, \mathrm{Enc}(s))$. Hence, a tree automaton reading $\mathrm{Enc}(s)$ is able to calculate for each position $d \in \mathrm{Enc}(s)$ the pairs of initial and final states of loops of $\mathrm{LStck}(d)$. As every run decomposes as a sequence of loops separated by a single operation, knowing $\mathrm{Loops}(s')$ for each $s' \leq s$ enables the automaton to check the correctness of a candidate for a certificate of reachability.

[4] We consider the final state reached by A on input w as the value it calculates for w.

As a tree-automaton may non-deterministically guess a certificate of the reachability of a configuration, the encodings of reachable configurations form a regular set.

5.3. Extension to Regular Reachability

By now, we have already established the tree-automaticity of each CPG G since we have seen that our encoding yields a regular image of the vertices of G and the transition relations are turned into regular relations of the tree encoding. Using similar techniques, we can improve this result:

Theorem 5.10. *If G is the ε-closure of some* CPG G' *then* (G, Reach) *is tree-automatic where* Reach *is the binary predicate that is true on a pair* (c_1, c_2) *of configurations if there is a path from c_1 to c_2 in G.*

Remark 5.11. Each graph in the second level of the Caucal-hierarchy can be obtained as the ε-contraction of some level 2 CPG (see [3]) whence all these graphs are tree-automatic.

For a CPS S let $R \subseteq \Delta^*$ be a regular language over the transitions of S. As collapsible pushdown graphs are closed under products with finite automata even the reachability predicate Reach_R with restriction to R is tree-automatic. Here, $\mathrm{Reach}_R xy$ holds if there is a path from x to y in CPG(S) that uses a sequence of transitions in R. If A is the automaton recognising R, we obtain that $\mathrm{Reach}_R(q,s)(q',s')$ holds in CPG(S) iff $\mathrm{Reach}\big((q,q_i),s\big)\big((q',q_f),s'\big)$ holds in CPG($S \times A$) where q_i is the initial and q_f the unique final state of A. Using this idea one can define a CPG G' which is basically CPG($S \cup (S \times A)$) extended by transitions from (q,s) to $((q,q_i),s)$ and to $((q,q_f),s)$. CPG(S) as well as Reach_R w.r.t. CPG(S) are FO[Reach]-interpretable in G'. Hence we obtain:

Theorem 5.12. *Given a collapsible pushdown graph of level* 2*, its* FO[Reach_R] *theory is decidable for each regular $R \subseteq \Delta^*$.*

5.4. Computation of concrete tree-automatic representations of CPG

Up to now, we have only seen that there is a tree-automatic representation for each CPG. For computing a concrete representation, we rely on the following lemma.

Lemma 5.13. *Given some* CPS $S = (\Gamma, Q, \Delta, q_0)$*, some $q \in Q$, and some stack s, it is decidable whether (q,s) is a vertex of* CPG(S)*.*

The proof is based on the idea that a stack is uniquely determined by its top element and the information which substacks can be reached via collapse- and pop_i-operations. Hence we can construct an extension S' of S and a modal formula $\varphi_{q,s}$ such that there is some element $v \in \mathrm{CPG}(S')$ satisfying $\mathrm{CPG}(S'), v \models \varphi_{q,s}$ iff $(q,s) \in \mathrm{CPG}(S)$. S' basically contains new states for every substack of s and connects the different states via the appropriate pop_i-operations which are only applied if the topmost symbol of the stack agrees with the symbol we would expect when starting the pop_i-sequence in configuration (q,s).

From this lemma we can derive the computability of the automata in Lemma 5.8. Having obtained these automata, the construction of a tree-automatic representation of some CPG is directly derived from the proofs yielding the following theorem.

Theorem 5.14. *There is an algorithm that, given a level* 2 CPG G *and regular sets* $R_1, \dots, R_n \subseteq \Delta^*$*, computes a tree-automatic representation of* $(G, \mathrm{Reach}_{R_1}, \dots, \mathrm{Reach}_{R_n})$*.*

6. Conclusion

We have seen that level 2 collapsible pushdown graphs are tree-automatic. This result holds also if we apply ε-contractions and if we add regular reachability predicates. This implies that the second level of the Caucal-hierarchy is tree-automatic. But our result can only be seen as a starting point for further investigations of the CPG hierarchy: are level 3 collapsible pushdown graphs tree-automatic? We know an example of a level 5 CPG which is not tree-automatic. But even when tree-automaticity of all CPG cannot be expected, the question remains whether all CPG have decidable FO theories. In order to solve this problem one has to come up with new techniques.

A rather general question concerning our result aims at our knowledge about tree-automatic structures. Recent developments in the string case [9] show the decidability of rather large extensions of first-order logic for automatic structures. It would be interesting to clarify the status of the analogous claims for tree-automatic structures. Positive answers concerning the decidability of extensions of first-order logic on tree-automatic structures would give us the corresponding decidability results for collapsible pushdown graphs of level 2.

References

[1] R. Alur, S. Chaudhuri, and P. Madhusudan. Languages of nested trees. In *Proc. 18th International Conference on Computer-Aided Verification*, volume 4144 of *LNCS*, pages 329–342. Springer, 2006.

[2] A. Blumensath. Automatic structures. Diploma thesis, RWTH Aachen, 1999.

[3] A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2003*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003.

[4] D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, pages 165–176, 2002.

[5] J. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970.

[6] M. Hague, A. S. Murawski, C-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS '08: Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461, 2008.

[7] A. Kartzow. FO model checking on nested pushdown trees. In *MFCS'09*, volume 5734 of *LNCS*, pages 451–463. Springer, 2009.

[8] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS'02*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.

[9] D. Kuske. Theories of automatic structures and their complexity. In *CAI'09, Third International Conference on Algebraic Informatics*, volume 5725 of *LNCS*, pages 81–98. Springer, 2009.

[10] A. N. Maslov. The hierarchy of indexed languages of an arbitrary level. *Sov. Math., Dokl.*, 15:1170–1174, 1974.

[11] A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12:38–43, 1976.

[12] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

APPROXIMATE SHORTEST PATHS AVOIDING A FAILED VERTEX : OPTIMAL SIZE DATA STRUCTURES FOR UNWEIGHTED GRAPHS

NEELESH KHANNA[1] AND SURENDER BASWANA[2]

[1] Oracle India Pvt. Ltd, Bangalore-560029, India.
E-mail address: neelesh.khanna@gmail.com

[2] Indian Institute of Technology Kanpur, India.
E-mail address: sbaswana@cse.iitk.ac.in

Abstract. Let $G = (V, E)$ be any undirected graph on V vertices and E edges. A path $\mathbf{P}$ between any two vertices $u, v \in V$ is said to be t-approximate shortest path if its length is at most t times the length of the shortest path between u and v. We consider the problem of building a compact data structure for a given graph G which is capable of answering the following query for any $u, v, z \in V$ and $t > 1$.

report t-approximate shortest path between u and v when vertex z fails

We present data structures for the single source as well all-pairs versions of this problem. Our data structures guarantee optimal query time. Most impressive feature of our data structures is that their size *nearly* match the size of their best static counterparts.

1. Introduction

The shortest paths problem is a classical and well studied algorithmic problem of computer science. This problem requires processing of a given graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges to compute a data structure using which shortest path or distance between any two vertices can be efficiently reported. Two famous and thoroughly studied versions of this problem are single source shortest paths (SSSP) problem and all-pairs shortest paths (APSP) problem.

Most of the applications of the shortest paths problem involve real life graphs and networks which are prone to failure of nodes (vertices) and links (edges). This has motivated researchers to design *dynamic* solution for the shortest paths problem. For this purpose, one has to first develop a suitable model for the shortest paths problem in dynamic networks. In fact two such models exists, and each of them has its own algorithmic objectives.

The shortest paths problem in the first model is described as follows : There is an initial graph followed by an on-line sequence of insertion and deletion of edges interspersed

1998 ACM Subject Classification: E.1 [**Data Structures**]:Graphs and Networks; G.2.2[**Discrete Mathematics**]:Graph Theory - *Graph Algorithms* .

Key words and phrases: Shortest path, distance, distance queries, oracle.

Part of this work was done while the authors were at Max-Planck Institute for Computer Science, Saarbruecken, Germany during the period May-July 2009.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2481

with shortest path (or distance) queries. Each query has to be answered with respect to the graph which exists at that moment (incorporating all the updates preceding the query on the initial graph). A trivial solution of this problem is to recompute all-pairs shortest paths from scratch after each update. This is certainly a wasteful approach since a single update usually does not cause a huge change in the all-pairs distance information. Therefore, the algorithmic objective here is to maintain a data structure which can answer distance query efficiently and can be updated after any edge insertion or deletion in an efficient manner. In particular, the time required to update the data structure has to be substantially less than the running time of the best static algorithm. Many novel algorithms have been designed in the last ten years for this problem and its variants (see [6] and the references therein).

On one hand the first model is important since it captures the worst possible hardness of any dynamic graph problem. On the other hand, it can also be considered as a pessimistic model for real life networks. It is true that the networks are never immune to failures. But in addition to it, it is also rare to have networks which may have arbitrary number of failures in normal circumstances. It is essential for network designers to choose suitable technology to make sure that the failures are quite infrequent in the network. In addition, when a vertex or edge fails (goes down), it does not remain failed/down indefinitely. Instead, it comes up after some finite time due to simultaneous repair mechanism going on in the network. These aspects can be captured in the second model which takes as input a graph and a number $\ell \ll n$. This model assumes that there will be at most ℓ vertices or edges which may be inactive at any time, though the corresponding set of failed vertices or edges may keep changing as the time progresses : the old failed vertices become active while some new active vertices may fail. The algorithmic objective in this model is to preprocess the given graph to construct a compact data structure which for any subset S of at most ℓ vertices may answer the following query for any $u, v \in V$.

Report the shortest-path (or distance) from u to v in $G\backslash S$.

It is desired that each query gets answered in *optimal time* : retrieval of distance in $O(1)$ time and the shortest path in time which is of the order of the number of its edges. The ultimate research goal would be to understand the complexity of the above problem for any given value ℓ. In this pursuit, the first natural step would be to efficiently solve and thoroughly understand the complexity of the problem for the case $\ell = 1$, that is, the shortest paths problem avoiding any failed vertex. Interestingly, this problem appears as a sub problem in many other related problems, namely, Vickrey pricing of networks [9], most vital node of a shortest path [11], the replacement path problem [12], and shortest paths avoiding forbidden subpaths [1].

The first nontrivial and quite significant breakthrough on the all-pairs version of this problem was made by Demetrescu et al. [7]. They designed an $O(n^2 \log n)$ space data structure, namely *distance sensitivity oracle*, which is capable of reporting the shortest path between any two vertices avoiding any single failed vertex. The preprocessing time of this data structure is $O(mn^2)$. Recently, Bernstein and Karger [4] improved the preprocessing time to $O(mn \log n)$. Though $\Theta(n^2 \log n)$ space bound of this all-pairs distance sensitivity oracle is optimal up to logarithmic factors, it is too large for many real life graphs which appear in various large scale applications [13]. In most of these graphs usually $m \ll n^2$, hence a table of $\Theta(n^2)$ size may be too large for practical purposes. However, it is also known [7] that even a data structure which reports exact distances from a fixed source avoiding a single failed vertex will require $\Omega(n^2)$ space in the worst case. So approximation seems to be the only way to design a small space compact data structure for the problem of shortest paths avoiding a failed vertex. A path between $u, v \in V$ is said to be t-approximate shortest path if its length is at most t times that of the shortest path between the two. The factor t is usually called the stretch. We would like to state here that many algorithms and data structures have been designed in the last fifteen years for the static all-pairs approximate shortest paths (see [2, 13] and references therein). The prime motivation underlying these algorithms has been to achieve sub-quadratic space and/or sub-cubic preprocessing time for the static APSP problem. However, no data structure was designed in the past for approximate shortest paths avoiding any failed vertex.

In this paper, we present really compact data structures which are capable of reporting approximate shortest paths between two vertices avoiding any failed vertex in undirected graphs. The most impressive feature of our data structures is their nearly optimal size. In fact their size almost matches the size of their best static counterparts.

1.1. New Results

Single source approximate shortest paths avoiding any failed vertex.
First we address weighted graphs. For the weighted graphs, we present an $O(m \log n)$ time constructible data structure of size $O(n \log n)$ which can report 3-approximate shortest path from the source to any vertex $v \in V$ avoiding any $x \in V$. We then consider the case of undirected unweighted graphs. For these graphs, we present an $O(n \frac{\log n}{\epsilon^3})$ space data structure which can even report $(1+\epsilon)$-approximate shortest path for any $\epsilon > 0$.
All-pairs approximate shortest paths avoiding any failed vertex.
Among the existing data structures for static all-pairs approximate shortest paths, the *approximate distance oracle* of Thorup and Zwick [13] stands out due to its amazing features. Thorup and Zwick [13] showed that an undirected graph can be preprocessed in sub-cubic time to build a data structure of size $O(kn^{1+1/k})$ for any $k > 1$. This data structure, despite of its sub-quadratic size, is capable of reporting $(2k-1)$-approximate distance between any two vertices in $O(k)$ time (and the corresponding approximate shortest path in optimal time), and hence the name *oracle*. Moreover, the size-stretch trade off achieved by this data structure is essentially optimal. It is a very natural question to explore whether it is possible to design all-pairs approximate distance oracle which may handle single vertex failure. We show that it is indeed possible for unweighted graphs. For this purpose, we suitably modify the approximate distance oracle of Thorup and Zwick [13] using some new insights and our single source data structure mentioned above. These modifications make the approximate shortest-paths oracle of Thorup and Zwick handle vertex failure easily, and (surprisingly) still preserving the old (optimal) trade-off between the space and the stretch. For precise details, see Theorem 5.3.

For the algorithmic details missing in this extended abstract due to page limitations, we suggest the reader to refer to the journal version [10]. Our data structures can be easily adapted for handling edge failure as well without any increase in space or time complexity.

2. Preliminaries

We use the following notations and definitions in the context of a given undirected graph $G = (V, E)$ with $n = |V|$, $m = |E|$ and a weight function $\omega : E \rightarrow \mathbf{R}^+$.

- T_r : single source shortest path tree rooted at r.
- $\mathbf{P}(x, y)$: the shortest path between x and y.

- $\delta(x, y)$: the length of the shortest path between x and y.
- $\mathbf{P}(x, y, z)$: the shortest path between x and y avoiding vertex z.
- $\delta(x, y, z)$: the length of the shortest path between x and y avoiding vertex z.
- $T_r(x)$: the subtree of T_r rooted at x.
- $G_r(x)$: the subgraph induced by the vertices of set $T_r(x)$ and augmented by vertex r and edges from r as follows. For each $v \in T_r(x)$ with neighbors outside $T_r(x)$, keep an edge (r, v) of weight $= \min_{(u,v)\in E, u\notin T_r(x)}(\delta(r, u) + \omega(u, v))$.
- $P :: Q$: a path formed by concatenating path Q at the end of path P with an edge $(u, v) \in E$, where u is the last vertex of P and v is the first vertex of Q.
- $E(X)$: the set of edges from E with at least one endpoint in X.

Our algorithms will also use a data structure for answering lowest common ancestor (LCA) queries on T_r. There exists an $O(n)$ time computable data structure which occupies $O(n)$ space and can answer any LCA query in $O(1)$ time (see [3] and references therein).

3. Single source 3-approximate shortest paths avoiding a failed vertex

We shall first solve a simpler sub-problem where the vertex which may fail belong to a given path $P \in T_r$. Then we use divide and conquer strategy wherein we decompose T_r into a set of disjoint paths and for each such path, we solve this sub-problem.

3.1. Solving the Sub-Problem : the failures of a vertex from a given path $\mathbf{P}(r, t)$

Given the shortest path tree T_r, let $\mathbf{P}(r, t) = \langle r(= x_0), x_1, ..., x_k(= t)\rangle$ be any shortest path present in T_r. We shall design an $O(n)$ space data structure which will support retrieval of a 3-approximate shortest path from r to any $v \in V$ when some vertex from $\mathbf{P}(r, t)$ fails. The preprocessing time of our algorithm will be $O(m + n \log n)$ which matches that of Dijkstra's algorithm. The algorithm is inspired by the algorithm of Nardelli et al. [11] for computing the most vital vertex on a shortest path. Consider vertex x_i lying on the path $\mathbf{P}(r, t)$. We partition the tree $T_r \backslash \{x_i\}$ into the following 3 parts (see Figure 1).

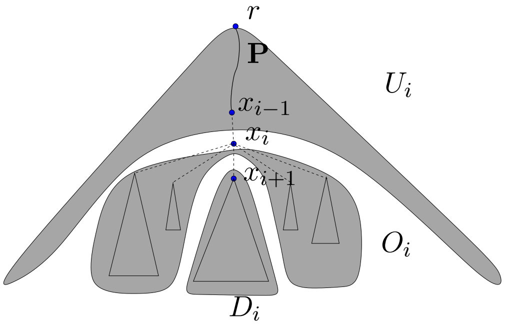


Figure 1: Partitioning of the shortest path tree T_r at $x_i \in \mathbf{P}$

1. U_i : the tree T_r after removing the subtree $T_r(x_i)$
2. D_i : the subtree of T_r rooted at x_{i+1}
3. O_i : the portion of T_r left after removing U_i, x_i, and D_i.

Note that a vertex of the tree T_r is either a vertex of the path $\mathbf{P}(r, t)$ or it belongs to some O_i for some i. We build the following two data-structures of total $O(n)$ size.

(1) a data structure to retrieve 3-approximate shortest path from r to any $v \in D_i$.
(2) a data structure to retrieve 3-approximate shortest path from r to any $v \in O_i$.

3.1.1. *Data structure for 3-approximate shortest paths to vertices of D_i when x_i has failed.* Consider the vertex x_{i+1} and any other vertex $y \in D_i$. Note that the shortest path $\mathbf{P}(x_{i+1}, y)$ remains intact even after removal of x_i, and its length is certainly less than $\delta(r, y)$. Based on this simple observation one can intuitively see that in order to travel from r to y when x_i fails, we may travel along shortest route to x_{i+1} (that is $\mathbf{P}(r, x_{i+1}, x_i)$) and then along $\mathbf{P}(x_{i+1}, y)$. Using triangle inequality and the fact that the graph is undirected, the length of this path $\mathbf{P}(r, x_{i+1}, x_i) :: \mathbf{P}(x_{i+1}, y)$ can be approximated as follows.

$$\begin{aligned}\delta(r, x_{i+1}, x_i) + \delta(x_{i+1}, y) &\leq \delta(r, y, x_i) + \delta(y, x_{i+1}, x_i) + \delta(x_{i+1}, y)\\ &\leq \delta(r, y, x_i) + 2\delta(x_{i+1}, y)\\ &\leq \delta(r, y, x_i) + 2\delta(r, y) \leq 3\delta(r, y, x_i)\end{aligned}$$

Therefore, in order to support retrieval of 3-approximate shortest path to any $v \in D_i$ in optimal time, it suffices to store the path $\mathbf{P}(r, x_{i+1}, x_i)$.

In order to devise ways of efficient computation and compact storage of $\mathbf{P}(r, x_{i+1}, x_i)$ for a given i, we use the following lemma about the structure of the path $\mathbf{P}(r, x_{i+1}, x_i)$.

Lemma 3.1. *The shortest path $\boldsymbol{P}(r, x_{i+1}, x_i)$ is of the form $P_1 :: P_2$ where P_1 is a shortest path from r in the subgraph induced by $U_i \cup O_i$, and P_2 is a path present in D_i.*

It follows that in order to compute $\mathbf{P}(r, x_{i+1}, x_i)$, first we need to compute shortest paths from r in the subgraph induced by $U_i \cup O_i$. Let $\delta_i(r, v)$ denote the distance from r to $v \in U_i \cup O_i$ in this subgraph. Note that $\delta_i(r, v)$ for $v \in U_i$ and the corresponding shortest path is the same as in the original graph, and is already present in T_r. For computing shortest paths from r to vertices of O_i, we build a shortest path tree (denoted as $T_r(O_i)$) from r in the subgraph induced by vertices $O_i \cup \{r\}$ and the following additional edges. For each $z \in O_i$ with at least one neighbor in U_i, we add an edge (r, z) with weight $= min_{(u,z) \in E, u \in U_i}(\delta(r, u) + \omega(u, z))$. Applying Lemma 3.1, let (y_i, z_i) be the edge of $\mathbf{P}(r, x_{i+1}, x_i)$ joining the sub path present in $U_i \cup O_i$ with the sub path present in D_i. This edge can be identified using the fact that this is the edge which minimizes $\delta_i(r, y) + \omega(y, z) + \delta(x_{i+1}, z)$ over all $z \in D_i, y \in U_i \cup O_i, (y, z) \in E$. The vertex x_{i+1} stores the path $\mathbf{P}(r, x_{i+1}, x_i)$ implicitly by keeping the edge (y_i, z_i) and the tree $T_r(O_i)$. The shortest path $\mathbf{P}(r, x_{i+1}, x_i)$ can be retrieved in optimal time using the trees T_r, $T_r(O_i)$, and the edge (y_{i+1}, z_{i+1}). Due to mutual disjointness of O_i's, the overall space requirement of the data structure for retrieving $\mathbf{P}(r, x_{i+1}, x_i)$ for all $i \leq k$ will be $O(n)$.

3.1.2. *Data structure for 3-approximate shortest paths to vertices of O_i when x_i has failed.* In order to compute 3-approximate shortest path to O_i upon failure of x_i, we shall use the approximate shortest paths to D_i as computed above. Here we use an interesting observation which states that if we have a data structure to retrieve α-approximate shortest paths from r to vertices of D_i when x_i fails, then we can use it to have a data-structure to retrieve α-approximate shortest paths to vertices of O_i as well. To prove this result, this is how we proceed. Consider the subgraph induced by O_i and augmented with vertex r and some extra edges which are defined as follows.

- For each $o \in O_i$ having neighbors from U_i, keep an edge (r, o) and assign it weight $= \min_{(u,o)\in E, u\in U_i}(\delta(r,u) + \omega(u,o))$.
- For each $o \in O_i$ having neighbors from D_i, keep an edge (r, o) and assign it weight $= \min_{(u,o)\in E, u\in D_i}(\hat{\delta}(r,u,x_i) + \omega(u,o))$, where $\hat{\delta}(r,u,x_i)$ is the α-approximate distance to u upon failure of x_i. (In the present situation we have $\alpha = 3$.)

Let us denote this graph as $G_r(O_i)$. Observation 3.3 is based on the following lemma which is easy to prove.

Lemma 3.2. *The Dijkstra's algorithm from r in the graph $G_r(O_i)$ computes α-approximate shortest paths from r to all $v \in O_i$ avoiding x_i.*

Observation 3.3. If we can design a data structure for retrieving $(1+\epsilon)$-approximate shortest paths from r to vertices of D_i upon failure of x_i, then it can also be used to design a data structure which can support retrieval of $(1+\epsilon)$-approximate shortest paths to all vertices of the graph upon failure of x_i.

We compute and store the shortest path tree rooted at r in the graph $G_r(O_i)$. This tree along with the tree T_r and the data structure described in the previous sub-section suffice for retrieval of 3-approximate shortest paths to $o \in O_i$ upon failure of x_i.

Query answering: Suppose the oracle receives a query asking for approximate shortest path from r to v avoiding $x_i \in \mathbf{P}(r,t)$. It first invokes lowest common ancestor (LCA) query between v and x_i on T_r. If $LCA(v, x_i) \neq x_i$, the shortest path from r to v remains unaffected and so it reports the path $\mathbf{P}(r,t)$. Otherwise, it determines if $v \in D_i$ or $v \in O_i$. Depending upon the two cases, it reports the approximate shortest path between r and v_i using one of the two data structures described above.

Theorem 3.4. *An undirected weighted graph $G = (V,E)$, a source $r \in V$, and a shortest path $P \in T_r$ can be processed in $O(m + n\log n)$ time to build a data structure of $O(n)$ space which can report 3-approximate shortest path from r to any $v \in V$ avoiding any single failed vertex from P.*

3.2. Handling the failure of any vertex in T_r

We follow divide and conquer strategy based on the following simple lemma.

Lemma 3.5. *There exists an $O(n)$ time algorithm to compute a path P in T_r whose removal splits T_r into a collection of disjoint subtrees $T_r(v_1), ...T_r(v_j)$ such that*

- *$|T_r(v_i)| < n/2$ for each $i \le j$.*
- *$P \cup_i T_r(v_i) = T$ and $P \cap T_r(v_i) = \emptyset\ \forall i$.*

First we compute the path $P \in T_r$ as mentioned in Lemma 3.5. We build the data structure for handling failure of any vertex from P by executing the algorithm of Theorem 3.4. Let $v_1, ..., v_j$ be the roots of the sub trees of T_r connected to the path P with an edge. For each $1 \le i \le j$, we solve the problem recursively on the subgraph $G_r(v_i)$, and build the corresponding data structures. Lemma 3.5 and Theorem 3.4 can be used in straight forward manner to prove the following theorem.

Theorem 3.6. *An undirected weighted graph $G = (V,E)$ can be processed in $O(m\log n + n\log^2 n)$ time to build a data structure of size $O(n\log n)$ which can answer, in optimal time, any 3-approximate shortest path query from a given source r to any vertex $v \in V$ avoiding any single failed vertex.*

4. Single source (1+ϵ)-approximate shortest paths avoiding a failed vertex

In this section, we shall present a compact data structure for single source $(1+\epsilon)$-approximate shortest paths avoiding a failed vertex in an unweighted graph. Let $level(v)$ denote the level (or distance from r) of vertex v in the tree T_r. Let U_x, D_x, O_x denote the partitions of the tree T_r formed by deletion of vertex x, with the same meaning as that of U_i, D_i, O_i defined for x_i in the previous section. On the basis of Observation 3.3, our objective is to build a compact data structure which will support retrieval of $(1+\epsilon)$-approximate shortest-paths to vertices of D_x upon failure of x for any $x \in V$. Let $\texttt{uchild}(x)$ denote the root of the subtree corresponding to D_x (it is similar to x_{i+1} in case of D_i). For reporting approximate distance between r and $v \in D_x$ when x fails, the data structure of previous section reports path of length $\delta(r, \texttt{uchild}(x), x) + \delta(\texttt{uchild}(x), v)$ which is bounded by $\delta(r, v, x) + 2\delta(\texttt{uchild}(x), v)$. It should be noted that the approximation factor associated with it is already bounded by $(1+\epsilon)$ for any $\epsilon > 0$ if the following condition holds.

$\mathbf{C}$: $\texttt{uchild}(x)$ is *close* to v, that is, $\delta(\texttt{uchild}(x), v) \leq \frac{\epsilon}{2}\delta(r, v)$.

We shall build a supplementary data structure which will ensure that whenever the condition $\mathbf{C}$ does not hold, there will be some ancestor w of v lying on $\mathbf{P}(x, v)$, called a *special* vertex, satisfying the following two properties.

(1) $\delta(w, v) \ll \delta(r, v)$, that is w is much closer to v than r.
(2) vertex w stores approximate shortest path to r avoiding x (with the approximation factor arbitrarily close to 1).

We shall refer to such vertices w as special-vertices.

4.1. Constructing the set of special vertices

Let h be the height of BFS tree rooted at r. Let L be a set of integers such that $L = \{i | \lfloor (1+\epsilon)^i \rfloor < h\}$. For a given $i \in L$, we define a subset S_i of special vertices as $S_i = \{u \in V | level(u) = \lfloor (1+\epsilon)^i \rfloor \wedge |T_r(u)| \geq \epsilon level(u)\}$. We define the set of special vertices as $S = \cup_{\forall i \in L} S_i$. In addition, we also introduce the following terminologies.

- $S(v)$: the nearest ancestor of v which belongs to set S.
- $V(u)$: For a vertex $u \in S$, $V(u)$ denotes the set of vertices $v \in V$ with $S(v) = u$. In essence, the vertex u will serve as the special vertex for each vertex from $V(u)$. For failure of any vertex $x \in \mathbf{P}(r, u)$, each vertex of set $V(u)$ will query the data structure stored at u for retrieval of approximate shortest path/distance from the source.

We now state two simple lemmas based on the above construction.

Lemma 4.1. *Let $v \in V \backslash S$, then $\delta(v, S(v)) \leq \left(\frac{2\epsilon}{1+\epsilon}\right) level(v)$ if $\epsilon < 1$*

Lemma 4.2. *Let u be a vertex at level ℓ and $u \in S$. Then $V(u) \geq \epsilon\ell$.*

If we can ensure that the data structure for a special vertex u (for retrieving approximate shortest paths from r upon failure of any $x \in \mathbf{P}(r, u)$) is of size $O(level(u))$, then it would follow from Lemma 4.2 that the space required by our supplementary data structure will be linear in n.

4.2. The data structure for a special vertex

Consider a special vertex v with $level(v) = \lfloor (1+\epsilon)^i \rfloor$ We shall now describe a compact data structure stored at v which will facilitate retrieval of approximate shortest path from r to v upon failure of any vertex $x \in \mathbf{P}(r, v)$.

Let v' be the special vertex which is present at level $\lfloor (1+\epsilon)^{i-1} \rfloor$ and is ancestor of v. The data structure stored at v will be defined in a way that will prevent it from storing information that is already present in the data structure of some special vertex lying on $\mathbf{P}(r, v')$.

If $x \in \mathbf{P}(v', v)$, then the data structure described in the previous section itself stores a path which is $(1 + 2\epsilon)$-approximation of $\mathbf{P}(r, v, x)$.

Let us now consider the nontrivial case when $x \in \mathbf{P}(r, v'), x \neq v'$. In order to discuss this case, we would like to introduce the notion of *detour*. To understand it, let us visualize the paths $\mathbf{P}(r, v, x)$ and $\mathbf{P}(r, v)$ simultaneously. Since $\mathbf{P}(r, v, x)$ and $\mathbf{P}(r, v)$ have the same end-points and x doesn't lie on $\mathbf{P}(r, v, x)$, there must be a *middle* portion of $\mathbf{P}(r, v, x)$ which intersects $\mathbf{P}(r, v)$ at exactly two vertices, and the remaining portion of $\mathbf{P}(r, v, x)$ overlaps with $\mathbf{P}(r, v)$. This *middle* portion is called a detour. We now define it more formally. Let a and b be two vertices on the shortest path $\mathbf{P}(r, v)$. We use $a \prec b$ to denote that vertex a is closer to r than vertex b. The notation $a \preceq b$ would mean that either $a \prec b$ or $a = b$. . So here is the definition of detour (and the underlying observation).

Definition 4.3. Let $x \in \mathbf{P}(r, y)$. When x fails, the path $\mathbf{P}(r, y, x)$ will be of the form of $\mathbf{P}(r, a) :: p_{a,b} :: \mathbf{P}(b, y)$, where $r \preceq a \prec x \prec b \preceq y$ and the path $p_{a,b}$ is such that $p_{a,b} \cap \mathbf{P}(a, b) = \{a, b\}$. In other words, $p_{a,b}$ meets $\mathbf{P}(a, b)$ only at the end points. We shall call $p_{a,b}$ as the detour associated with the shortest path $\mathbf{P}(r, y, x)$.

Let $p_{a,b}$ represent the detour w.r.t. to $\mathbf{P}(r, v, x)$. The handling of failure of vertices $x \in \mathbf{P}(r, v)$ which lie above v' would depend upon the detour $p_{a,b}$. This detour can be of any of the following types (see Figure 2 for illustration).

- I : $b \preceq v'$.
- II : $v' \prec b$.

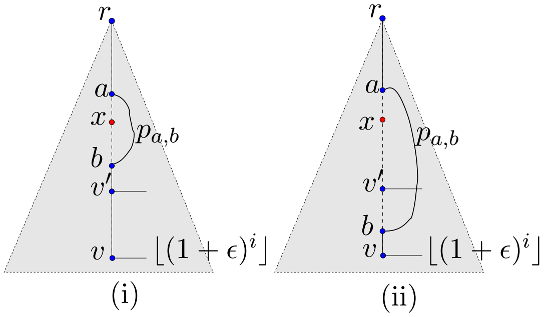

r
a
x
$p_{a,b}$
b
v'
v
$\lfloor (1+\epsilon)^i \rfloor$
(i)
r
a
x
$p_{a,b}$
v'
b
v
$\lfloor (1+\epsilon)^i \rfloor$
(ii)

Figure 2: $p_{a,b}$ is shortest detour of $\mathbf{P}(r, v, x)$. (i) : detour of type I, (ii) : detour of type II

Handling detours of type I is relatively easy. Let w be the farthest ancestor of v such that $w \in S$ and level of w is greater or equal to the level of b. In this case, v stores the corresponding detour implicitly by just keeping a pointer to the vertex w.

Handling detours of type II is slightly tricky since we can't afford to store each of them explicitly. However, we shall employ the following observation associated with the detours of type II to guarantee low space requirement.

Observation 4.4. Let α_1, α_2,...,α_t be the vertices on $\mathbf{P}(r,v)$ (in the increasing order of their levels) such that the shortest detour corresponding to $\mathbf{P}(r,v,\alpha_i)$ is of type II $\forall i$, then

$$\delta(r,v,\alpha_1) \geq \delta(r,v,\alpha_2) \geq \cdots \geq \delta(r,v,\alpha_t)$$

It follows from the above observation that if $\delta(r,v,\alpha_i) \leq (1+\epsilon)\delta(r,v,\alpha_j)$ for any $i<j$, then $\mathbf{P}(r,v,\alpha_i)$ may as well serve as $(1+\epsilon)$-approximate shortest path from r to v avoiding α_j. In other words, we need not store the detour associated with $\mathbf{P}(r,v,\alpha_j)$ in such situation. Using this observation, we shall have to explicitly store only $O(\log_{1+\epsilon} n)$ detours of type II. Moreover, we do not store explicitly detours of type II whose length is much larger than $level(v)$. Specifically, if $\mathbf{P}(r,v,x) \geq \frac{1}{\epsilon} level(v)$, then v will merely store pointer to the path $\mathbf{P}(r, \texttt{uchild}(x), x) :: \mathbf{P}(\texttt{uchild}(x), v)$. This ensures that each detour of type II which v has to explicitly store will have length $O(\frac{1}{\epsilon} level(v))$.

It follows from the above description that for a special vertex v and $x \in \mathbf{P}(r,v)$, the data structure associated with v stores $(1+2\epsilon)$-approximation of the path $\mathbf{P}(r,v,x)$. Moreover, the total space required by the data structure associated with all the special vertices will be $O(n\frac{\log n}{\epsilon^3})$. This supplementary data structure combined with the data structure of previous section can report $(1+6\epsilon)$-approximation of $\mathbf{P}(r,z,x)$ for any $z,x \in V$.

Theorem 4.5. *Given an undirected unweighted graph $G=(V,E)$, source $r \in V$, and any $\epsilon > 0$, we can build a data structure of size $O(n\frac{\log n}{\epsilon^3})$ that can report $(1+\epsilon)$-approximate shortest path from r to any $z \in V$ avoiding any failed vertex in optimal time.*

5. All-pairs $(2k-1)(1+\epsilon)$-approx. distance oracle avoiding a failed vertex

We start with a brief description of the approximate distance oracle of Thorup and Zwick [13]. The key idea to achieve sub-quadratic space is to store distance from each vertex to only a small set of vertices. For retrieving approximate distance between any two vertices $u,v \in V$, it is ensured that there is a third vertex w which is *close* to both of them, and whose distance from both of them is known. To realize this idea, Thorup and Zwick [13] introduced two novel structures called *ball* and *cluster* which are defined for any two subsets A,B of vertices as follows. (here $\delta(v,B)$ denotes the distance between v and its nearest vertex from B).

$$Ball(v,A,B) = \{w \in A | \delta(v,w) < \delta(v,B)\} \qquad C(w,A,B) = \{v \in V | \delta(v,w) < \delta(v,B)\}$$

Construction of $(2k-1)$-approximate distance oracle of Thorup and Zwick [13] employs a k-level hierarchy $\mathbf{A}_k = \langle A_0 \supseteq A_1 \supseteq A_2 ... \supseteq A_{k-1} \supset A_k \rangle$ of subsets of vertices as follows.
$A_0 = V$, $A_k = \emptyset$, and A_{i+1} for any $i < k-1$ is formed by selecting each vertex from A_i independently with probability $n^{-1/k}$.

The data structure associated with the $(2k-1)$-approximate distance oracle of Thorup and Zwick [13] stores for each vertex $v \in V$ the following information :

- the vertices of set $\cup_{i<k} Ball(v,A_i,A_{i+1})$ (and their distances).
- the vertex from A_i nearest to v (to be denoted as $p_i(v)$).

Due to randomization underlying the construction of $\mathbf{A}_k$, the expected size of $Ball(v,A_i,A_{i+1})$ is $O(n^{1/k})$, and hence the space required by the oracle is $O(kn^{1+1/k})$. We shall now outline the ideas in extending the $(2k-1)$-approximate distance oracle to handle single vertex failure. Kindly refer to the extended version [10] of this paper for complete details.

5.1. Overview of all-pairs approx. distance oracles avoiding a failed vertex

Firstly the notations used by the static approximate distance oracle of [13], in particular ball and cluster, get extended for single vertex failure in a natural manner as follows. (here $\delta(v, B, x)$ is the distance between v and its nearest vertex from B in $G\backslash\{x\}$).

$$Ball^x(v, A, B) = \{w \in A | \delta(v, w, x) < \delta(v, B, x)\}$$

$$C^x(w, A, B) = \{v \in V | \delta(v, w, x) < \delta(v, B, x)\}$$

Let $p_i^x(v)$ denote the vertex from A_i which is nearest to v in $G\backslash\{x\}$. Along the lines of the static approximate distance oracle of Thorup and Zwick [13], the basic operation which the approximate distance oracle avoiding a failed vertex should support is the following :

Report distance (exact or approximate) between v and $w \in A_i$ if $w \in Ball^x(v, A_i, A_{i+1})$ for any given $v, x \in V$.

However, it can be observed that we would have to support this operation implicitly instead of explicitly keeping $Ball^x(v, A_i, A_{i+1})$ for each v, x, i. Our starting point is the simple observation that clusters and balls are inverses of each others, that is, $w \in Ball^x(v, A_i, A_{i+1})$ is equivalent to $v \in C^x(w, A_i, A_{i+1})$. Now we make an important observation. Consider the subgraph $\mathbf{G}_i(w)$ induced by the vertices of set $\cup_{x \in V} C^x(w, A_i, A_{i+1})$. This subgraph preserves the path $\mathbf{P}(w, v, x)$ for each $x, v \in V$ if $w \in Ball^x(v, A_i, A_{i+1})$. So it suffices to keep a single source (approximate) shortest paths oracle on $\mathbf{G}_i(w)$ with w as the root. Keeping this data structure for each $w \in A_i$ provides an implicit compact data structure for supporting the basic operation mentioned above. Using Theorem 4.5, it can be seen that the space required at a level i will be of the order of $\sum_{w \in A_i} | \cup_{x \in V} C^x(w, A_i, A_{i+1})|$, but it is not clear whether we can get an upper bound of the order of $n^{1+1/k}$ on this quantity. Here, as a new tool, we introduce the notion of ϵ-truncated balls and clusters.

Definition 5.1. Given a vertex x, any subsets A, B, and $\epsilon > 0$

$$Ball^x(v, A, B, \epsilon) = \left\{ w \in A | \delta(v, w, x) < \frac{\delta(v, B, x)}{1 + \epsilon} \right\}$$

Instead of dealing with the usual balls (and clusters) under deletion of single vertex, we deal with ϵ-truncated balls (and clusters) under deletion of single vertex. We note that the inverse relationship between clusters and balls gets seamlessly extended to ϵ-truncated balls and clusters under single vertex failure as well. That is,

$$\sum_{w \in A_i} | \cup_{x \in V} C^x(w, A_i, A_{i+1}, \epsilon)| = \sum_{v \in V} | \cup_{x \in V} Ball^x(v, A_i, A_{i+1}, \epsilon)|$$

So it suffices to get an upper bound on the size of the set $\cup_{x \in V} Ball^x(v, A_i, A_{i+1}, \epsilon)$ for any vertex $v \in V$. The following lemma states a very crucial property of ϵ-truncated balls which leads to prove the existence of a small set S of $O(\frac{1}{\epsilon^2} \log n)$ vertices such that

$$\cup_{x \in V} Ball^x(v, A_i, A_{i+1}, \epsilon) \subseteq \cup_{x \in S} Ball^x(v, A_i, A_{i+1}) \cup Ball(v, A_i, A_{i+1}) \tag{5.1}$$

Lemma 5.2. *In a given graph $G = (V, E)$, let v be any vertex and let $u = p_{i+1}(v)$. Let x_1 and x_2 be any two vertices on the $\boldsymbol{P}(v, u)$ path with x_1 appearing closer to v on this path and $\delta(v, A_{i+1}, x_1) \leq (1 + \epsilon)\delta(v, A_{i+1}, x_2)$. Then*

$$Ball^{x_1}(v, A_i, A_{i+1}, \epsilon) \subseteq Ball(v, A_i, A_{i+1}) \cup Ball^{x_2}(v, A_i, A_{i+1})$$

Proof. Let w be any vertex in A_i. It suffices to show the following. If w does not belong to $Ball(v, A_i, A_{i+1}) \cup Ball^{x_2}(v, A_i, A_{i+1})$, then w does not belong to $Ball^{x_1}(v, A_i, A_{i+1}, \epsilon)$. The proof is based on the analysis of the following two cases.

Case 1 : The vertex x_2 is present in $\mathbf{P}(v, w, x_1)$.

Since, $w \notin Ball(v, A_i, A_{i+1})$, therefore, $\delta(v, w)$ is at least $\delta(v, u)$. Hence using triangle inequality, $\delta(v, x_2) + \delta(x_2, w) \geq \delta(v, u)$. Now $\delta(v, u) = \delta(v, x_2) + \delta(x_2, u)$ (since x_2 lies on $P(v, u)$). Hence $\delta(x_2, w) \geq \delta(x_2, u)$. Moreover, since x_1 does not appear on $\mathbf{P}(x_2, u)$, so $\delta(x_2, u) = \delta(x_2, u, x_1)$. So

$$\delta(x_2, w, x_1) \geq \delta(x_2, u, x_1) \tag{5.2}$$

Now it is given that $x_2 \in \mathbf{P}(v, w, x_1)$, so $\mathbf{P}(v, w, x_1)$ must be of the form $\mathbf{P}(v, x_2, x_1) :: \mathbf{P}(x_2, w, x_1)$, the length of which is at least $\delta(v, x_2, x_1) + \delta(x_2, u, x_1)$ using Equation 5.2. The latter quantity is at least $\delta(v, u, x_1)$ which by definition is at least $\delta(v, A_{i+1}, x_1)$. Hence $w \notin Ball^{x_1}(v, A_i, A_{i+1})$, and therefore, $w \notin Ball^{x_1}(v, A_i, A_{i+1}, \epsilon)$.

Case 2 : The vertex x_2 is not present in $\mathbf{P}(v, w, x_1)$.

In this case, $\delta(v, w, x_1) = \delta(v, w, \{x_1, x_2\}) \geq \delta(v, w, x_2)$. The value $\delta(v, w, x_2)$ is in turn at least $\delta(v, A_{i+1}, x_2)$ since $w \notin Ball^{x_2}(v, A_i, A_{i+1})$. It is given that $\delta(v, A_{i+1}, x_2) \geq \frac{\delta(v, A_{i+1}, x_1)}{1+\epsilon}$, hence conclude that $\delta(v, w, x_1) \geq \frac{\delta(v, A_{i+1}, x_1)}{1+\epsilon}$. So $w \notin Ball^{x_1}(v, A_i, A_{i+1}, \epsilon)$. ■

We shall now outline the construction of a small set S of vertices which will satisfy Equation 5.1. Let $u = p_{i+1}(v)$ and let $\mathbf{P}(v, u) = v(= x_0), x_1, ..., x_\ell(= u)$. Observe that $\cup_{x \in V} Ball^x(v, A_i, A_{i+1}, \epsilon) = \cup_{1 \leq j \leq \ell} Ball^{x_j}(v, A_i, A_{i+1}, \epsilon)$. For any node $x \in \mathbf{P}(u, v)$, let $value(x) = \delta(v, A_{i+1}, x)$, and let h be the maximum *value* of any node on this path. The set S is initially empty.

Let $\alpha(1)$ be the largest index from $[1, \ell]$ such that $value(x_i) \geq h/(1 + \epsilon)$. It can be seen that for all $j < \alpha(1)$, $\delta(v, A_{i+1}, x_j) \leq (1 + \epsilon)\delta(v, A_{i+1}, x_{\alpha(1)})$. Therefore, it follows from Lemma 5.2 that for each vertex $x \in \{x_1, ..., x_{\alpha(1)}\}$, $Ball^x(v, A_i, A_{i+1}, \epsilon) \subseteq Ball^{x_{\alpha(1)}}(v, A_i, A_{i+1}) \cup Ball(v, A_i, A_{i+1})$. So we insert $x_{\alpha(1)}$ to S. Similarly $\alpha(2) \in [\alpha(1) + 1, \ell]$ be the greatest integer such that $value(x_{\alpha(2)}) \geq h/(1 + \epsilon)^2$. We add $x_{\alpha(2)}$ to S, and so on. It can be seen that the set S constructed in this manner will satisfy Equation 5.1 and its size will be $O(\log_{1+\epsilon} h) = O(\frac{\log n}{\epsilon})$.

It can be shown using elementary probability theory that for each $x \in V$, the set $Ball^x(v, A_i, A_{i+1})$ has size $O(n^{1/k} \log n)$ with high probability. Therefore, the construction of the set S outlined above implies the following crucial bound for each $v \in V, i < k - 1$ which helps us design all-pairs approximate distance oracle avoiding a failed vertex.

$$| \cup_{x \in V} Ball^x(v, A_i, A_{i+1}, \epsilon)| = O\left(n^{1/k} \frac{\log^2 n}{\epsilon}\right)$$

Using this equation, and owing to inverse relationship between clusters and balls, it follows that $\sum_{w \in A_i} | \cup_{x \in V} C^x(w, A_i, A_{i+1}, \epsilon)| = O\left(n^{1+1/k} \frac{\log^2 n}{\epsilon}\right)$. Our all-pairs approximate distance oracle avoiding any failed vertex will keep the following data structures.

- Let $p_i^x(v, \epsilon)$ denote a vertex w from A_i with $\delta(v, w, x) \leq (1+\epsilon)\delta(v, p_i^x(v), x)$. We keep a data structure $\mathbf{N}_i$ $\forall i < k$, using which we can retrieve $p_i^x(v, \epsilon)$. This data-structure is obtained by suitable augmentation of our single source $(1+\epsilon)$-approximate oracle.
- For each $w \in A_i$, we keep our single source $(1 + \epsilon)$-approximate oracle in $\mathbf{G}_i(w, \epsilon)$ which is the subgraph induced by $\cup_{x \in V} C^x(w, A_i, A_{i+1}, \epsilon)$.

It follows that the overall space required by the data structure will be $O(kn^{1+1/k}\frac{\log^3 n}{\epsilon^4})$. The query algorithm and the analysis on the stretch of the approximate distance reported by the oracle are similar in spirit to that of Thorup and Zwick [13] (see [10] for details).

Theorem 5.3. *Given an integer $k > 1$ and a fraction $\epsilon > 0$, an unweighted graph $G = (V, E)$ can be processed to construct a data structure which can answer $(2k-1)(1+\epsilon)$-approximate distance query between any two nodes $u \in V$ and $v \in V$ avoiding any single failed vertex in $O(k)$ time. The total size of the data structure is $O(kn^{1+1/k}\frac{\log^3 n}{\epsilon^4})$.*

Future work. (i) Can we design a data structure for single source $(1+\epsilon)$-approx. shortest paths avoiding a failed vertex for weighted graphs ? Such a data structure will immediately extend our all-pairs approx. distance oracle avoiding a failed vertex to weighted graphs.
(ii) How to design approx. distance oracles avoiding two or more failed vertices ? Recent work of Duan and Pettie [8], and Chechik et al. [5] provides additional motivation for this.

References

[1] M. Ahmed and A. Lubiw. Shortest paths avoiding forbidden subpaths. In *STACS '09: Proceedings of 26th International Symposium on Theoretical Aspects of Computer Science*, pages 63–74, Freiburg, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[2] S. Baswana and T. Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 591–602, Washington, DC, USA, 2006. IEEE Computer Society.

[3] M. A. Bender and M. Farach-Colton. The lca problem revisited. In *LATIN '00: Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, pages 88–94, London, UK, 2000. Springer-Verlag.

[4] A. Bernstein and D. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 101–110, New York, NY, USA, 2009. ACM.

[5] S. Chechik, M. Langberg, D. Peleg, and L. Roditty. Fault-tolerant spanners for general graphs. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 435–444, New York, NY, USA, 2009. ACM.

[6] C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004.

[7] C. Demetrescu, M. Thorup, R. A. Chowdhury, and V. Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

[8] R. Duan and S. Pettie. Dual-failure distance and connectivity oracles. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 506–515, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[9] J. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 252, Washington, DC, USA, 2001. IEEE Computer Society.

[10] N. Khanna and S. Baswana. Approximate shortest paths avoiding a failed vertex : optimal data structures for unweighted graphs. *http://www.cse.iitk.ac.in/∼sbaswana/publications/algorithmica-09.pdf.*

[11] E. Nardelli, G. Proietti, and P. Widmayer. Finding the most vital node of a shortest path. *Theor. Comput. Sci.*, 296(1):167–177, 2003.

[12] L. Roditty. On the k-simple shortest paths problem in weighted directed graphs. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 920–928, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[13] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 525-536
www.stacs-conf.org

HOLANT PROBLEMS FOR REGULAR GRAPHS WITH COMPLEX EDGE FUNCTIONS

MICHAEL KOWALCZYK[1] AND JIN-YI CAI[2]

[1] Department of Mathematics and Computer Science, Northern Michigan University
Marquette, MI 49855, USA
E-mail address: mkowalcz@nmu.edu

[2] Computer Sciences Department, University of Wisconsin, Madison, WI 53706, USA
E-mail address: jyc@cs.wisc.edu

Abstract. We prove a complexity dichotomy theorem for Holant Problems on 3-regular graphs with an arbitrary complex-valued edge function. Three new techniques are introduced: (1) higher dimensional iterations in interpolation; (2) Eigenvalue Shifted Pairs, which allow us to prove that a pair of combinatorial gadgets *in combination* succeed in proving #P-hardness; and (3) algebraic symmetrization, which significantly lowers the *symbolic complexity* of the proof for computational complexity. With *holographic reductions* the classification theorem also applies to problems beyond the basic model.

1. Introduction

In this paper we consider the following subclass of Holant Problems [5, 6]. An input regular graph $G = (V, E)$ is given, where every $e \in E$ is labeled with a (symmetric) edge function g. The function g takes 0-1 inputs from its incident nodes and outputs arbitrary values in $\mathbb{C}$. The problem is to compute the quantity $\text{Holant}(G) = \sum_{\sigma:V\to\{0,1\}} \prod_{\{u,v\}\in E} g(\{\sigma(u), \sigma(v)\})$.

Holant Problems are a natural class of counting problems. As introduced in [5, 6], the general Holant Problem framework can encode all Counting Constraint Satisfaction Problems (#CSP). This includes special cases such as weighted Vertex Cover, Graph Colorings, Matchings, and Perfect Matchings. The subclass of Holant Problems in this paper can also be considered as (weighted) H-homomorphism (or H-coloring) problems [2, 3, 7, 8, 9, 10] with an arbitrary 2×2 symmetric complex matrix H, however *restricted to* regular graphs G as input. E.g., Vertex Cover is the case when $H = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$. When the matrix H is a 0-1 matrix, it is called unweighted. Dichotomy theorems (i.e., the problem is either in P or #P-hard, depending on H) for unweighted H-homomorphisms with undirected graphs H and directed acyclic graphs H are given in [8] and [7] respectively. A

1998 ACM Subject Classification: F.2.1.
Key words and phrases: Computational complexity.
The second author is supported by NSF CCF-0830488 and CCF-0914969.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2482

dichotomy theorem for any symmetric matrix H with non-negative real entries is proved in [2]. Goldberg et al. [9] proved a dichotomy theorem for all real symmetric matrices H. Finally, Cai, Chen, and Lu have proved a dichotomy theorem for all complex symmetric matrices H [3].

The crucial difference between Holant Problems and #CSP is that in #CSP, Equality functions of arbitrary arity are *presumed* to be present. In terms of H-homomorphism problems, this means that the input graph is allowed to have vertices of arbitrarily high degrees. This may appear to be a minor distinction; in fact it has a major impact on complexity. It turns out that if Equality gates of arbitrary arity are freely available in possible inputs then it is technically easier to prove #P-hardness. Proofs of previous dichotomy theorems make extensive use of constructions called thickening and stretching. These constructions require the availability of Equality gates of arbitrary arity (equivalently, vertices of arbitrarily high degrees) to carry out. Proving #P-hardness becomes more challenging in the degree restricted case. Furthermore there are indeed cases within this class of counting problems where the problem is #P-hard for general graphs, but solvable in P when restricted to 3-regular graphs.

We denote the (symmetric) edge function g by $[x, y, z]$, where $x = g(0,0)$, $y = g(0,1) = g(1,0)$ and $z = g(1,1)$. Functions will also be called gates or signatures. (For Vertex Cover, the function corresponding to H is the Or gate, and is denoted by the signature $[0,1,1]$.) In this paper we give a dichotomy theorem for the complexity of Holant Problems on 3-regular graphs with arbitrary signature $g = [x, y, z]$, where $x, y, z \in \mathbb{C}$. First, if $y = 0$, the Holant Problem is easily solvable in P. Assuming $y \neq 0$ we may normalize g and assume $y = 1$. Our main theorem is as follows:

Theorem 1.1. *Suppose $a, b \in \mathbb{C}$, and let $X = ab$, $Z = (\frac{a^3+b^3}{2})^2$. Then the Holant Problem on 3-regular graphs with $g = [a, 1, b]$ is #P-hard except in the following cases, for which the problem is in P.*

(1) $X = 1$.
(2) $X = Z = 0$.
(3) $X = -1$ *and* $Z = 0$.
(4) $X = -1$ *and* $Z = -1$.

If we restrict the input to planar 3-regular graphs, then these four categories are solvable in P*, as well as a fifth category $X^3 = Z$. The problem remains #P-hard in all other cases.* [1] ■

These results can be extended to k-regular graphs (we detail how this is accomplished in a forthcoming work). One can also use holographic reductions [15] to extend this theorem to more general Holant Problems.

In order to achieve this result, some new proof techniques are introduced. To discuss this we first take a look at some previous results. Valiant [13, 14] introduced the powerful technique of *interpolation*, which was further developed by many others. In [5] a dichotomy theorem is proved for the case when g is a Boolean function. The technique from [5] is to provide certain algebraic criteria which ensure that *interpolation* succeeds, and then apply these criteria to prove that (a large number yet) finitely many individual problems are #P-hard. This involves (a small number of) gadget constructions, and the algebraic criteria

[1] Technically, computational complexity involving complex or real numbers should, in the Turing model, be restricted to computable numbers. In other models such as the Blum-Shub-Smale model [1] no such restrictions are needed. Our results are not sensitive to the exact model of computation.

are powerful enough to show that they succeed in each case. Nonetheless this involves a case-by-case verification. In [6] this theorem is extended to all real-valued a and b, and we have to deal with infinitely many problems. So instead of focusing on one problem, we devised (a large number of) recursive gadgets and analyzed the regions of $(a, b) \in \mathbb{R}^2$ where they fail to prove #P-hardness. The algebraic criteria from [5] are not suitable (Galois theoretic) for general a and b, and so we formulated weaker but simpler criteria. Using these criteria, the analysis of the failure set becomes expressible as containment of semi-algebraic sets. As semi-algebraic sets are decidable, this offers the ultimate possibility that *if* we found enough gadgets to prove #P-hardness, *then* there is a *computational* proof (of computational intractability) in a finite number of steps. However this turned out to be a tremendous undertaking in symbolic computation, and many additional ideas were needed to finally carry out this plan. In particular, it would seem hopeless to extend that approach to all complex a and b.

In this paper, we introduce three new ideas. (1) We introduce a method to construct gadgets that carry out iterations at a higher dimension, and then collapse to a lower dimension for the purpose of constructing unary signatures. This involves a starter gadget, a recursive iteration gadget, and a finisher gadget. We prove a lemma that guarantees that among polynomially many iterations, some subset of them satisfies properties sufficient for interpolation to succeed (it may not be known *a priori* which subset worked, but that does not matter). (2) Eigenvalue Shifted Pairs are coupled pairs of gadgets whose transition matrices differ by δI where $\delta \neq 0$. They have shifted eigenvalues, and by analyzing their failure conditions, we can show that except on very rare points, one or the other gadget succeeds. (3) Algebraic symmetrization. We derive a new expression of the Holant polynomial over 3-regular graphs, with a crucially reduced degree. This simplification of the Holant and related polynomials condenses the problem of proving #P-hardness to the point where all remaining cases can be handled by symbolic computation. We also use the same expression to prove tractability.

The rest of this paper is organized as follows. In Section 2 we discuss notation and background information. In Section 3 we cover interpolation techniques, including how to collapse higher dimensional iterations to interpolate unary signatures. In Section 4 we show how to perform algebraic symmetrization of the Holant, and introduce Eigenvalue Shifted Pairs (ESP) of gadgets. Then we combine the new techniques to prove Theorem 1.1. Some proofs are omitted due to space limitations; a full version will appear in [11].

2. Notations and Background

We state the counting framework more formally. A *signature grid* $\Omega = (G, \mathcal{F}, \pi)$ consists of a labeled graph $G = (V, E)$ where π labels each vertex $v \in V$ with a function $f_v \in \mathcal{F}$. We consider all edge assignments $\xi : E \to \{0, 1\}$; f_v takes inputs from its incident edges $E(v)$ at v and outputs values in $\mathbb{C}$. The counting problem on the instance Ω is to compute[2]

$$\text{Holant}_\Omega = \sum_{\xi} \prod_{v \in V} f_v(\xi \mid_{E(v)}).$$

Suppose G is a bipartite graph (U, V, E) such that each $u \in U$ has degree 2. Furthermore suppose each $v \in V$ is labeled by an EQUALITY gate $=_k$ where $k = \deg(v)$. Then any non-zero term in Holant_Ω corresponds to a 0-1 assignment $\sigma : V \to \{0, 1\}$. In fact, we can merge

[2] The term Holant was first introduced by Valiant in [15] to denote a related exponential sum.

the two incident edges at $u \in U$ into one edge e_u, and label this edge e_u by the function f_u. This gives an edge-labeled graph (V, E') where $E' = \{e_u : u \in U\}$. For an edge-labeled graph (V, E') where $e \in E'$ has label g_e, $\text{Holant}_\Omega = \sum_{\sigma:V\to\{0,1\}} \prod_{e=(v,w)\in E'} g_e(\sigma(v), \sigma(w))$. If each g_e is the same function g (but assignments $\sigma : V \to [q]$ take values in a finite set $[q]$) this is exactly the H-coloring problem (for undirected graphs g is a symmetric function). In particular, if (U, V, E) is a $(2, k)$-regular bipartite graph, equivalently $G' = (V, E')$ is a k-regular graph, then this is the H-coloring problem restricted to k-regular graphs. In this paper we will discuss 3-regular graphs, where each g_e is the same symmetric complex-valued function. We also remark that for general bipartite graphs (U, V, E), giving EQUALITY (of various arities) to all vertices on one side V defines #CSP as a special case of Holant Problems. But whether EQUALITY of various arities are present has a major impact on complexity, thus Holant Problems are a refinement of #CSP.

A symmetric function $g : \{0, 1\}^k \to \mathbb{C}$ can be denoted as $[g_0, g_1, \ldots, g_k]$, where g_i is the value of g on inputs of Hamming weight i. They are also called *signatures*. Frequently we will revert back to the bipartite view: for $(2, 3)$-regular bipartite graphs (U, V, E), if every $u \in U$ is labeled $g = [g_0, g_1, g_2]$ and every $v \in V$ is labeled $r = [r_0, r_1, r_2, r_3]$, then we also use $\#[g_0, g_1, g_2] \mid [r_0, r_1, r_2, r_3]$ to denote the Holant Problem. Note that $[1, 0, 1]$ and $[1, 0, 0, 1]$ are EQUALITY gates $=_2$ and $=_3$ respectively, and the main dichotomy theorem in this paper is about $\#[x, y, z] \mid [1, 0, 0, 1]$, for all $x, y, z \in \mathbb{C}$. We will also denote $\text{Hol}(a, b) = \#[a, 1, b] \mid [1, 0, 0, 1]$. More generally, If $\mathcal{G}$ and $\mathcal{R}$ are sets of signatures, and vertices of U (resp. V) are labeled by signatures from $\mathcal{G}$ (resp. $\mathcal{R}$), then we also use $\#\mathcal{G} \mid \mathcal{R}$ to denote the bipartite Holant Problem. Signatures in $\mathcal{G}$ are called *generators* and signatures in $\mathcal{R}$ are called *recognizers*. This notation is particularly convenient when we perform holographic transformations. Throughout this paper, all $(2, 3)$-regular bipartite graphs are arranged with generators on the degree 2 side and recognizers on the degree 3 side.

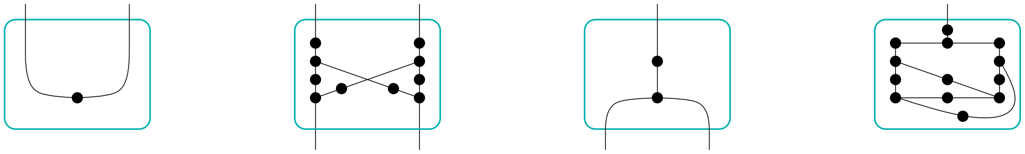
We use Arg to denote the principal value of the complex argument; i.e., $\text{Arg}(c) \in (-\pi, \pi]$ for all nonzero $c \in \mathbb{C}$.

2.1. $\mathcal{F}$-Gate

Any signature from $\mathcal{F}$ is available at a vertex as part of an input graph. Instead of a single vertex, we can use graph fragments to generalize this notion. An $\mathcal{F}$-gate Γ is a pair $(H, \mathcal{F})$, where $H = (V, E, D)$ is a graph with some dangling edges D (Figure 1 contains some examples). Other than these dangling edges, an $\mathcal{F}$-gate is the same as a signature grid. The role of dangling edges is similar to that of external nodes in Valiant's notion [16], however we allow more than one dangling edge for a node. In $H = (V, E, D)$ each node is assigned a function in $\mathcal{F}$ (we do not consider "dangling" leaf nodes at the end of a dangling edge among these), E are the regular edges, and D are the dangling edges. Then we can define a function for this $\mathcal{F}$-gate $\Gamma = (H, \mathcal{F})$,

$$\Gamma(y_1, y_2, \ldots, y_q) = \sum_{(x_1,x_2,\ldots,x_p)\in\{0,1\}^p} H(x_1, x_2, \ldots, x_p, y_1, y_2, \ldots, y_q),$$

where $p = |E|$, $q = |D|$, $(y_1, y_2, \ldots, y_q) \in \{0, 1\}^q$ denotes an assignment on the dangling edges, and $H(x_1, x_2, \ldots, x_p, y_1, y_2, \ldots, y_q)$ denotes the value of the signature grid on an assignment of all edges, i.e., the product of evaluations at every vertex of H, for $(x_1, x_2, \ldots, x_p, y_1, y_2, \ldots, y_q) \in \{0, 1\}^{p+q}$. We will also call this function the signature of the



(a) A starter gadget (b) A recursive gadget (c) A finisher gadget (d) A planar embedding of a single iteration

Figure 1: Examples of binary starter, recursive, and finisher gadgets

$\mathcal{F}$-gate Γ. An $\mathcal{F}$-gate can be used in a signature grid as if it is just a single node with the same signature. We note that even for a very simple signature set $\mathcal{F}$, the signatures for all $\mathcal{F}$-gates can be quite complicated and expressive. Matchgate signatures are an example [16].

The dangling edges of an $\mathcal{F}$-gate are considered as input or output variables. Any m-input n-output $\mathcal{F}$-gate can be viewed as a 2^n by 2^m matrix M which transforms arity-m signatures into arity-n signatures (this is true even if m or n are 0). Our construction will transform symmetric signatures to symmetric signatures. This implies that there exists an equivalent $n+1$ by $m+1$ matrix $\widetilde{M}$ which operates directly on column vectors written in symmetric signature notation. We will henceforth identify the matrix $\widetilde{M}$ with the $\mathcal{F}$-gate itself. The constructions in this paper are based upon three different types of bipartite $\mathcal{F}$-gates which we call *starter gadgets*, *recursive gadgets*, and *finisher gadgets*. An *arity-r starter gadget* is an $\mathcal{F}$-gate with no input but r output edges. If an $\mathcal{F}$-gate has r input and r output edges then it is called an *arity-r recursive gadget*. Finally, an $\mathcal{F}$-gate is an *arity-r finisher gadget* if it has r input edges 1 output edge. As a matter of convention, we consider any dangling edge incident with a generator as an output edge and any dangling edge incident with a recognizer as an input edge; see Figure 1.

3. Interpolation Techniques

3.1. Binary recursive construction

In this section, we develop our new technique of higher dimensional iterations for interpolation of unary signatures.

Lemma 3.1. *Suppose $M \in \mathbb{C}^{3\times 3}$ is a nonsingular matrix, $s \in \mathbb{C}^3$ is a nonzero vector, and for all integers $k \geq 1$, s is not a column eigenvector of M^k. Let $F_i \in \mathbb{C}^{2\times 3}$ be three matrices, where* $\operatorname{rank}(F_i) = 2$ *for $1 \leq i \leq 3$, and the intersection of the row spaces of F_i is trivial $\{0\}$. Then for every n, there exists some $F \in \{F_i : 1 \leq i \leq 3\}$, and some $S \subseteq \{FM^k s : 0 \leq k \leq n^3\}$, such that $|S| \geq n$ and vectors in S are* pairwise *linearly independent.*

Proof. Let $k > j \geq 0$ be integers. Then $M^k s$ and $M^j s$ are nonzero and also linearly independent, since otherwise s is an eigenvector of M^{k-j}. Let $N = [M^j s, M^k s] \in \mathbb{C}^{3\times 2}$, then $\operatorname{rank}(N) = 2$, and $\ker(N^{\mathrm{T}})$ is a 1-dimensional linear subspace. It follows that there exists an $F \in \{F_i : 1 \leq i \leq 3\}$ such that the row space of F does not contain $\ker(N^{\mathrm{T}})$, and hence has trivial intersection with $\ker(N^{\mathrm{T}})$. In other words, $\ker(N^{\mathrm{T}} F^{\mathrm{T}}) = \{0\}$. We conclude that $FN \in \mathbb{C}^{2\times 2}$ has rank 2, and $FM^j s$ and $FM^k s$ are linearly independent.

Each F_i, where $1 \leq i \leq 3$, defines a coloring of the set $K = \{0, 1, \dots, n^3\}$ as follows: color $k \in K$ with the linear subspace spanned by $F_i M^k s$. Thus, F_i defines an equivalence relation $\approx_i$ where $k \approx_i k'$ iff they receive the same color. Assume for a contradiction that for each F_i, where $1 \leq i \leq 3$, there are not n pairwise linearly independent vectors among $\{F_i M^k s : k \in K\}$. Then, including possibly the 0-dimensional space $\{0\}$, there can be at most n distinct colors assigned by F_i. By the pigeonhole principle, some k and k' with $0 \leq k < k' \leq n^3$ must receive the same color for all F_i, where $1 \leq i \leq 3$. This is a contradiction and we are done. ∎

The next lemma says that under suitable conditions we can construct all unary signatures $[x, y]$. The method will be interpolation at a higher dimensional iteration, and finishing up with a suitable *finisher* gadget. The crucial new technique here is that when iterating at a higher dimension, we can guarantee the existence of *one* finisher gadget that succeeds on polynomially many steps, which results in overall success. Different finisher gadgets may work for different initial signatures and different input size n, but these need not be known in advance and have no impact on the final success of the reduction.

Lemma 3.2. *Suppose that the following gadgets can be built using complex-valued signatures from a finite generator set $\mathcal{G}$ and a finite recognizer set $\mathcal{R}$.*

(1) *A binary starter gadget with nonzero signature $[z_0, z_1, z_2]$.*

(2) *A binary recursive gadget with nonsingular recurrence matrix M, for which $[z_0, z_1, z_2]^{\mathrm{T}}$ is not a column eigenvector of M^k for any positive integer k.*

(3) *Three binary finisher gadgets with rank 2 matrices $F_1, F_2, F_3 \in \mathbb{C}^{2\times 3}$, where the intersection of the row spaces of F_1, F_2, and F_3 is the zero vector.*

Then for any $x, y \in \mathbb{C}$, $\#\mathcal{G} \cup \{[x, y]\} \mid \mathcal{R} \leq_T \#\mathcal{G} \mid \mathcal{R}$.

Proof. The construction begins with the binary starter gadget with signature $[z_0, z_1, z_2]$, which we call N_0. Let $\mathcal{F} = \mathcal{G} \cup \mathcal{R}$. Recursively, $\mathcal{F}$-gate N_{k+1} is defined to be N_k connected to the binary recursive gadget in such a way that the input edges of the binary recursive gadget are merged with the output edges of N_k. Then $\mathcal{F}$-gate G_k is defined to be N_k connected to one of the finisher gadgets, with the input edges of the finisher gadget merged with the output edges of N_k (see Figure 1(d)). Herein we analyze the construction with respect to a given bipartite signature grid Ω for the Holant Problem $\#\mathcal{G} \cup \{[x, y]\} \mid \mathcal{R}$, with underlying graph $G = (V, E)$. Let $Q \subseteq V$ be the set of vertices with $[x, y]$ signatures, and let $n = |Q|$. By Lemma 3.1 fix j so that at least $n + 2$ of the first $(n + 2)^3 + 1$ vectors of the form $F_j M^k [z_0, z_1, z_2]^{\mathrm{T}}$ are pairwise linearly independent. We use finisher gadget F_j in the recursive construction, so that the signature of G_k is $F_j M^k [z_0, z_1, z_2]^{\mathrm{T}}$, which we denote by $[X_k, Y_k]$. We note that there exists a subset S of these signatures for which each Y_k is nonzero and $|S| = n + 1$. We will argue using only the existence of S, so there is no need to algorithmically "find" such a set, and for that matter, one can try out all three finisher gadgets without any need to determine which finisher gadget is "the correct one" beforehand. If we replace every element of Q with a copy of G_k, we obtain an instance of $\#\mathcal{G} \mid \mathcal{R}$ (note that the correct bipartite signature structure is preserved), and we denote this new signature grid by Ω_k. Then

$$\mathrm{Holant}_{\Omega_k} = \sum_{0 \leq i \leq n} c_i X_k^i Y_k^{n-i}$$

where $c_i = \sum_{\sigma \in J_i} \prod_{v \in V \setminus Q} f_v(\sigma|_{E(v)})$, J_i is the set of $\{0, 1\}$ edge assignments where the number of 0s assigned to the edges incident to the copies of G_k is i, f_v is the signature at v, and $E(v)$ is the set of edges incident to v. The important point is that the c_i values do not depend on X_k or Y_k. Since each signature grid Ω_k is an instance of $\#\mathcal{G} \mid \mathcal{R}$, Holant_{Ω_k} can be solved exactly using the oracle. Carrying out this process for every $k \in \{0, 1, \ldots, (n+2)^3\}$, we arrive at a linear system where the c_i values are the unknowns.

$$\begin{bmatrix} \text{Holant}_{\Omega_0} \\ \text{Holant}_{\Omega_1} \\ \vdots \\ \text{Holant}_{\Omega_{(n+2)^3}} \end{bmatrix} = \begin{bmatrix} X_0^0 Y_0^n & X_0^1 Y_0^{n-1} & \cdots & X_0^n Y_0^0 \\ X_1^0 Y_1^n & X_1^1 Y_1^{n-1} & \cdots & X_1^n Y_1^0 \\ \vdots & \vdots & \ddots & \vdots \\ X_{(n+2)^3}^0 Y_{(n+2)^3}^n & X_{(n+2)^3}^1 Y_{(n+2)^3}^{n-1} & \cdots & X_{(n+2)^3}^n Y_{(n+2)^3}^0 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}.$$

Define $x_i = X_{k_i}$ and $y_i = Y_{k_i}$ where $S = \{k_0, k_1, \ldots, k_n\}$, so that $[x_i, y_i] \in S$ for $0 \leq i \leq n$, and we have a subsystem

$$\begin{bmatrix} y_0^{-n} \cdot \text{Holant}_{\Omega_0} \\ y_1^{-n} \cdot \text{Holant}_{\Omega_1} \\ \vdots \\ y_n^{-n} \cdot \text{Holant}_{\Omega_n} \end{bmatrix} = \begin{bmatrix} x_0^0 y_0^0 & x_0^1 y_0^{-1} & \cdots & x_0^n y_0^{-n} \\ x_1^0 y_1^0 & x_1^1 y_1^{-1} & \cdots & x_1^n y_1^{-n} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^0 y_n^0 & x_n^1 y_n^{-1} & \cdots & x_n^n y_n^{-n} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}.$$

The matrix above has entry $(x_r/y_r)^c$ at index (r, c). Due to pairwise linear independence of $[x_r, y_r]$, x_r/y_r is pairwise distinct for each $r \in S$. Hence this is a Vandermonde system of full rank. Therefore the initial feasible linear system has full rank and we can solve it for the c_i values. With these values in hand, we can calculate $\text{Holant}_\Omega = \sum_{0 \leq i \leq n} c_i x^i y^{n-i}$ directly, completing the reduction. ■

The ability to simulate all unary signatures will allow us to prove #P-hardness. The next lemma says that, if $\mathcal{R}$ contains the EQUALITY gate $=_3$, then other than on a 1-dimensional curve $ab = 1$ and an isolated point $(a, b) = (0, 0)$, the ability to simulate unary signatures gives a reduction from VERTEX COVER. Note that counting VERTEX COVER on 3-regular graphs is just $\#[0, 1, 1] \mid [1, 0, 0, 1]$. Xia et al. showed that this is #P-hard even when the input is restricted to 3-regular planar graphs [17]. We will see shortly that on the curve $ab = 1$ and at $(a, b) = (0, 0)$, the problem $\text{Hol}(a, b)$ is tractable.

Lemma 3.3. *Suppose that* $(a, b) \in \mathbb{C}^2 - \{(a, b) : ab = 1\} - \{(0, 0)\}$ *and let* $\mathcal{G}$ *and* $\mathcal{R}$ *be finite signature sets where* $[a, 1, b] \in \mathcal{G}$ *and* $[1, 0, 0, 1] \in \mathcal{R}$. *Further assume that* $\#\mathcal{G} \cup \{[x_i, y_i] : 0 \leq i < m\} \mid \mathcal{R} \leq_T \#\mathcal{G} \mid \mathcal{R}$ *for any* $x_i, y_i \in \mathbb{C}$ *and* $m \in \mathbb{Z}^+$. *Then* $\#\mathcal{G} \cup \{[0, 1, 1]\} \mid \mathcal{R} \leq_T \#\mathcal{G} \mid \mathcal{R}$, *and* $\#\mathcal{G} \mid \mathcal{R}$ *is* #P*-hard.*

Proof. Since $\#[0, 1, 1] \mid [1, 0, 0, 1]$ is #P-hard, we only need to show how to simulate the generator signature $[0, 1, 1]$. Respectively, Gadgets 1, 2, and 3 (Figure 2) can be used to simulate generator signatures $[b^{-1}, 1, 2b]$, $[0, 1, 5/(2a)]$, and $[0, 1, 1]$ in the cases where $ab = 0$, $ab = -1$, and both $ab \neq 0$ and $ab \neq -1$ (when $ab = 0$, we assume without loss of generality that $a = 0$ and $b \neq 0$). To carry this out, we set $\theta = [b, b^{-1}]$ in Gadget 1; $\theta = [1/(6a), -a/24]$ and $\gamma = [-3/a, a]$ in Gadget 2; and $\theta = (ab+1)(1-ab)^{-1}[1, -a^2]$, $\gamma = [-a^{-2}, b^{-1}(1+ab)^{-1}]$, and $\rho = (ab - 1)^{-1}[-b, a]$ in Gadget 3. This results in a chain of reductions to simulate $[0, 1, 1]$ in all cases (i.e. Gadget 2 simulates a signature to be used with Gadget 1, which in turn simulates a signature to be used with Gadget 3, and Gadget 3 simulates $[0, 1, 1]$). ■

It will be helpful to have conditions that are easier to check than those in Lemma 3.2. To this end, we establish condition 2 in terms of eigenvalues, and we build general-purpose finisher gadgets to eliminate condition 3. Let M_4, M_5, and F be the recurrence matrices for Gadget 4, Gadget 5, and the simplest possible binary finisher gadget (each

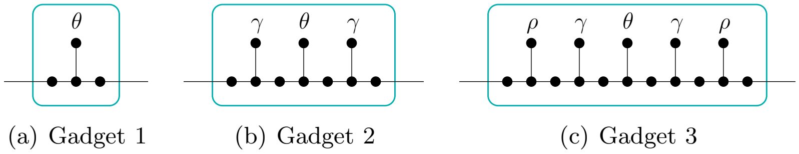


Figure 2: Gadgets used to simulate the [0,1,1] signature

built using generator signature $[a, 1, b]$ and recognizer signature $[1, 0, 0, 1]$; see Figures 3(a), 3(b), and 1(c)). Provided that $ab \neq 1$ and $a^3 \neq b^3$, it turns out that the finisher gadget sets $\{F, FM_4, FM_4^2\}$ and $\{F, FM_4, FM_5\}$ satisfy condition 3 of Lemma 3.2 when $ab \neq 0$ and $ab = 0$, respectively. Together with Lemma 3.3, these observations yield the following.

Theorem 3.4. *If the following gadgets can be built using generator* $[a, 1, b]$ *and recognizer* $[1, 0, 0, 1]$ *where* $a, b \in \mathbb{C}$, $ab \neq 1$, *and* $a^3 \neq b^3$, *then the problem* $\text{Hol}(a, b)$ *is* #P*-hard.*

(1) *A binary recursive gadget with nonsingular recurrence matrix* M *which has eigenvalues* α *and* β *such that* $\frac{\alpha}{\beta}$ *is not a root of unity.*

(2) *A binary starter gadget with signature* s *which is not orthogonal to any row eigenvector of* M*.* ■

3.2. Unary recursive construction

Now we consider the unary case. The following lemma arrives from [12] and is stated explicitly in [6]. It can be viewed as a unary version of Lemma 3.2 without finisher gadgets.

Lemma 3.5. *Suppose there is a unary recursive gadget with nonsingular matrix* M *and a unary starter gadget with nonzero signature vector* s*. If the ratio of the eigenvalues of* M *is not a root of unity and* s *is not a column eigenvector of* M*, then these gadgets can be used to interpolate all unary signatures.* ■

Surprisingly, a set of general-purpose starter gadgets can be made for this construction as long as $ab \neq 1$ and $a^3 \neq b^3$, so we refine this lemma by eliminating the starter gadget requirement. The starter gadgets are Fs, FM_4s, and FM_6s where M_6 is Gadget 6 and s is the single-vertex starter gadget (see Figures 3(c) and 1(a)).

Theorem 3.6. *Suppose there is a unary recursive gadget with nonsingular matrix* M*, and the ratio of the eigenvalues of* M *is not a root of unity. Then for any* $a, b \in \mathbb{C}$ *where* $ab \neq 1$ *and* $a^3 \neq b^3$*, there is a starter gadget built using generator* $[a, 1, b]$ *and recognizer* $[1, 0, 0, 1]$ *for which the resulting construction can be used to interpolate all unary signatures.* ■

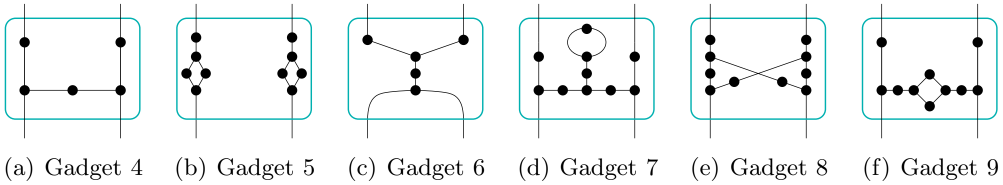


Figure 3: Binary recursive gadgets

4. Complex Signatures

Now we aim to characterize $\mathrm{Hol}(a, b)$ where $a, b \in \mathbb{C}$. The next lemma introduces the technique of algebraic symmetrization. We show that over 3-regular graphs, the Holant value is expressible as an integer polynomial $P(X, Y)$, where $X = ab$ and $Y = a^3 + b^3$. This change of variable, from (a, b) to (X, Y), is crucial in two ways. First, it allows us to derive tractability results easily, drawing connections between problems that may appear unrelated, and the tractability of one implies the other. Second, it facilitates the proof of hardness for those (a, b) where the problem is indeed #P-hard by reducing the degree of the polynomials involved. Once this transformation is made, four binary recursive gadgets easily cover all of the #P-hard problems where X and Y are real-valued, with a straightforward symbolic computation using CylindricalDecomposition in Mathematica$^{\mathrm{TM}}$. All gadget constructions in this section use $[a, 1, b]$ and $[1, 0, 0, 1]$ signatures exclusively, and we henceforth denote $X = ab$ and $Y = a^3 + b^3$ for the remainder of this paper.

Lemma 4.1. *Let G be a 3-regular graph. Then there exists a polynomial $P(\cdot, \cdot)$ with two variables and integer coefficients such that for any signature grid Ω having underlying graph G and every edge labeled $[a, 1, b]$, the Holant value is $\mathrm{Holant}_\Omega = P(ab, a^3 + b^3)$.*

Proof. Consider any $\{0, 1\}$ vertex assignment σ with a non-zero valuation. If σ' is the complement assignment switching all 0's and 1's in σ, then for σ and σ', we have the sum of valuations $a^i b^j + a^j b^i$ for some i and j. Here i (resp. j) is the number of edges connecting two degree 3 vertices both assigned 0 (resp. 1) by σ. We note that $a^i b^j + a^j b^i = (ab)^{\min(i,j)}(a^{|i-j|} + b^{|i-j|})$.

We prove $i \equiv j \pmod 3$ inductively. For the all-0 assignment, this is clear since every edge contributes a factor a and the number of edges is divisible by 3 for a 3-regular graph. Now starting from any assignment σ, if we switch the assignment on one vertex from 0 to 1, it is easy to verify that it changes the valuation from $a^i b^j$ to $a^{i'} b^{j'}$, where $i - j = i' - j' + 3$. As every $\{0, 1\}$ assignment is obtainable from the all-0 assignment by a sequence of switches, the conclusion $i \equiv j \pmod 3$ follows.

Now $a^i b^j + a^j b^i = (ab)^{\min(i,j)}(a^{3k} + b^{3k})$, for some $k \geq 0$ and a simple induction $a^{3(k+1)} + b^{3(k+1)} = (a^{3k} + b^{3k})(a^3 + b^3) - (ab)^3(a^{3(k-1)} + b^{3(k-1)})$ shows that the Holant is a polynomial $P(ab, a^3 + b^3)$ with integer coefficients. ∎

Corollary 4.2. *If $ab = -1$ and $a^{12} = 1$, then $\mathrm{Hol}(a, b)$ is in* P.

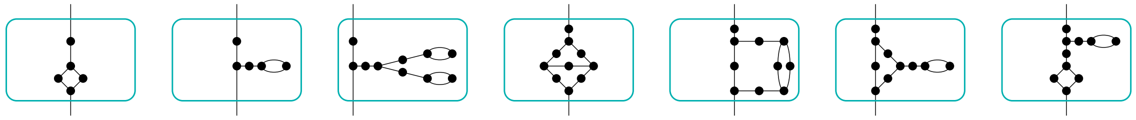
Proof. Immediate from Lemma 4.1, since the problems $\mathrm{Hol}(1, -1)$, $\mathrm{Hol}(-\mathfrak{i}, -\mathfrak{i})$, and $\mathrm{Hol}(\mathfrak{i}, \mathfrak{i})$ are all known to be solvable in P (these fall within the families $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}_3$ in [3]). ∎

We now list all of the cases where $\mathrm{Hol}(a, b)$ is computable in polynomial time.

Theorem 4.3. *If any of the following four conditions is true, then $\mathrm{Hol}(a, b)$ is solvable in* P*:*

(1) $ab = 1$,
(2) $a = b = 0$,
(3) $a^{12} = 1$ *and* $b = -a^{-1}$,
(4) $a^3 = b^3$ *and the input is restricted to planar graphs.*

Proof. If $ab = 1$ then the signature $[a, 1, b]$ is degenerate and the Holant can be computed in polynomial time. If $a = b = 0$, a 2-coloring algorithm can be employed on the edges. If $a^{12} = 1$ and $b = -a^{-1}$ then we are done by Corollary 4.2. If we restrict the input to planar graphs and $a^3 = b^3$, holographic algorithms can be applied [4]. ∎



(a) Gadget 10 (b) Gadget 11 (c) Gadget 12 (d) Gadget 13 (e) Gadget 14 (f) Gadget 15 (g) Gadget 16

Figure 4: Unary recursive gadgets

Our main task in this paper is to prove that all remaining problems are #P-hard. The following two lemmas provide sufficient conditions to satisfy the eigenvalue requirement of the recursive constructions.

Lemma 4.4. *If both roots of the complex polynomial $x^2 + Bx + C$ have the same norm, then $B|C| = \overline{B}C$ and $B^2\overline{C} = \overline{B}^2 C$. If further $B \neq 0$ and $C \neq 0$, then* $\operatorname{Arg}(B^2) = \operatorname{Arg}(C)$. ■

Lemma 4.5. *If all roots of the complex polynomial $x^3 + Bx^2 + Cx + D$ have the same norm, then $C|C|^2 = \overline{B}|B|^2 D$.* ■

Now we introduce a powerful new technique called *Eigenvalue Shifted Pairs.*

Definition 4.6. A pair of nonsingular square matrices M and M' is called an *Eigenvalue Shifted Pair (ESP)* if $M' = M + \delta I$ for some non-zero $\delta \in \mathbb{C}$, and M has distinct eigenvalues.

Clearly for such a pair, M' also has distinct eigenvalues. The recurrence matrices of Gadgets 10 and 11 (Figure 4) differ only by $ab-1$ along the diagonal, and form an Eigenvalue Shifted Pair for nearly all $a, b \in \mathbb{C}$. We will make significant use of such Eigenvalue Shifted Pairs, but first we state a technical lemma.

Lemma 4.7. *Suppose $\alpha, \beta, \delta \in \mathbb{C}$, $|\alpha| = |\beta|$, $\alpha \neq \beta$, $\delta \neq 0$, and $|\alpha + \delta| = |\beta + \delta|$. Then there exists $r, s \in \mathbb{R}$ such that $r\delta = \alpha + \beta$ and $s\delta^2 = \alpha\beta$.* ■

Corollary 4.8. *Let M and M' be an Eigenvalue Shifted Pair of 2 by 2 matrices. If both M and M' have eigenvalues of equal norm, then there exists $r, s \in \mathbb{R}$ such that* $\operatorname{tr}(M) = r\delta$ *(possibly 0) and* $\det(M) = s\delta^2$.

Proof. Let α and β be the eigenvalues of M, so $\alpha + \delta$ and $\beta + \delta$ are the eigenvalues of M'. Suppose that $|\alpha| = |\beta|$ and $|\alpha + \delta| = |\beta + \delta|$. Then by Lemma 4.7, there exists $r, s \in \mathbb{R}$ such that $\operatorname{tr}(M) = \alpha + \beta = r\delta$ and $\det(M) = \alpha\beta = s\delta^2$. ■

We now apply an ESP to prove that most settings of $\operatorname{Hol}(a, b)$ are #P-hard.

Lemma 4.9. *Suppose $X \neq \pm 1$, $X^2 + X + Y \neq 0$, and $4(X-1)^2(X+1) \neq (Y+2)^2$. Then either unary Gadget 10 or unary Gadget 11 has nonzero eigenvalues with distinct norm, unless X and Y are both real numbers.*

Proof. Gadgets 10 and 11 have $M_{10} = \begin{bmatrix} a^3+1 & a+b^2 \\ a^2+b & b^3+1 \end{bmatrix}$ and $M_{11} = \begin{bmatrix} a^3+ab & a+b^2 \\ a^2+b & ab+b^3 \end{bmatrix}$ as their recurrence matrices, so $M_{11} = M_{10} + (X-1)I$, and the eigenvalue shift is nonzero. Checking the determinants, $\det(M_{10}) = (X-1)^2(X+1) \neq 0$ and $\det(M_{11}) = (X-1)(X^2 + X + Y) \neq 0$. Also, $\operatorname{tr}(M_{10})^2 - 4\det(M_{10}) = (Y+2)^2 - 4(X-1)^2(X+1) \neq 0$, so the eigenvalues of M_{10} are distinct. Therefore by Corollary 4.8, either M_{10} or M_{11} has nonzero eigenvalues of distinct norm unless $\operatorname{tr}(M_{10}) = r(X-1)$ and $\det(M_{10}) = s(X-1)^2$ for some $r, s \in \mathbb{R}$. Then we would have $(X-1)^2(X+1) = s(X-1)^2$ so $X = s - 1 \in \mathbb{R}$ and $Y + 2 = r(X-1)$ so $Y = r(X-1) - 2 \in \mathbb{R}$. ■

Now we will deal with the following exceptional cases from Lemma 4.9 ($X = 1$ is tractable by Theorem 4.3).

0. $X \in \mathbb{R}$ and $Y \in \mathbb{R}$
1. $X^2 + X + Y = 0$
2. $X = -1$
3. $4(X-1)^2(X+1) = (Y+2)^2$

The case where X and Y are both real is dealt with using the tools developed in Section 3, and some symbolic computation. This includes the case where a and b are both real as a subcase. When a and b are both real, a dichotomy theorem for the complexity of $\mathrm{Hol}(a, b)$ has been proved in [6] with a significant effort. With the new tools developed, we offer a simpler proof. This also covers some cases where a or b is complex. Working with real-valued X and Y is a significant advantage, since the failure condition given by Lemma 4.5 is simplified by the disappearance of norms and conjugates. This brings the problem of proving #P-hardness within reach of symbolic computation via cylindrical decomposition. We apply Theorem 3.4 to Gadgets 4, 7, 8, and 9 (Figure 3) together with a starter gadget (Figure 1(a)) to prove that these problems are #P-hard. Conditions 1 and 2 of Theorem 3.4 are encoded directly into a query for CylindricalDecomposition in Mathematica$^{\mathrm{TM}}$.

Theorem 4.10. *Suppose $a, b \in \mathbb{C}$, $X, Y \in \mathbb{R}$, $ab \neq 1$, $a^3 \neq b^3$, and it is not the case that $a^6 = 1$ and $ab = -1$. Then the problem* $\mathrm{Hol}(a, b)$ *is* #P*-hard.* ■

Now we can assume that $X \notin \mathbb{R}$ or $Y \notin \mathbb{R}$, and we deal with the remaining three conditions. Note that if $X^2 + X + Y = 0$ then $X \in \mathbb{R}$ implies $Y \in \mathbb{R}$. So in the following lemma, the assumption that X and Y are not both real numbers amounts to $X \notin \mathbb{R}$.

Lemma 4.11. *If $X^2 + X + Y = 0$ and $X \notin \mathbb{R}$ then the recurrence matrix of unary Gadget 12 has nonzero eigenvalues with distinct norm.*

Proof. Let M_{12} be the recurrence matrix for unary Gadget 12. Then $\det(M_{12}) = X^6 - 6X^5 - X^4Y + 16X^4 + 11X^3Y - 10X^3 + 5X^2Y^2 - 7X^2Y - X^2 + XY^3 - 4XY^2 - 3XY - Y^3 - Y^2$. *Amazingly*, with the condition $X^2 + X + Y = 0$, this polynomial factors into $-X^2(X-1)^5$. Similarly, the trace, which is $-2X^3 + 6X^2 + 3XY + Y^2 + Y$, also factors into $X(X-1)^3$. Since $\det(M_{12}) \neq 0$, $\mathrm{tr}(M_{12}) \neq 0$, and $(1 - X)\det(M_{12}) = \mathrm{tr}(M_{12})^2$, we know $\mathrm{Arg}(\det(M_{12})) \neq \mathrm{Arg}(\mathrm{tr}(M_{12})^2)$ and conclude by Lemma 4.4 that the eigenvalues of M_{12} (which are nonzero) have distinct norm. ■

Similarly, Gadgets 11 and 13 can be used to deal with the $X = -1$ condition. The $4(X-1)^2(X+1) = (Y+2)^2$ condition can be dealt with using Gadgets 13 and 14 (an ESP) in addition to Gadgets 15 and 16. These gadgets, together with Lemma 3.3 and Theorem 3.6, give the following theorem (note that $X = -1$ and $Y = \pm 2\mathfrak{i}$ if and only if $a^3 = \pm\mathfrak{i}$ and $b = -a^{-1}$; any such setting of a and b is tractable by Theorem 4.3).

Theorem 4.12. *Suppose $a, b \in \mathbb{C}$ such that X and Y are not both real, $X \neq 1$, $a^3 \neq b^3$, and either $X \neq -1$ or $Y \neq \pm 2\mathfrak{i}$. Then the problem* $\mathrm{Hol}(a, b)$ *is* #P*-hard.* ■

Recall Vertex Cover is #P-hard on 3-regular planar graphs, and note that all gadgets discussed are planar (in the case of Gadget 8, each iteration can be redrawn in a planar way by "going around" the previous iterations; see Figure 1(d)). Thus, all of the hardness results proved so far still apply when the input graphs are restricted to planar graphs. There are, however, a few cases where the problem is #P-hard in general, yet is polynomial time computable when restricted to planar graphs. The relevant interpolation results can be obtained with Gadget 4 and holographic reductions, using a technique demonstrated in [5]. Given this, we have the following result.

Theorem 4.13. *The problem* $\mathrm{Hol}(a, b)$ *is* #P*-hard for all* $a, b \in \mathbb{C}$ *except in the following cases, for which the problem is in* P*.*

(1) $ab = 1$
(2) $a = b = 0$
(3) $a^{12} = 1$ *and* $b = -a^{-1}$

If we restrict the input to planar graphs, then these three categories are tractable in P*, as well as a fourth category* $a^3 = b^3$*, and the problem remains* #P*-hard in all other cases.* ■

A simple coordinate change from (a, b) to $(X, (\frac{Y}{2})^2)$ translates this into Theorem 1.1.

References

[1] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[2] A. Bulatov and M. Grohe. The complexity of partition functions. *Theoretical Computer Science*, 348(2-3):148–186, 2005.

[3] J-Y. Cai, X. Chen, and P. Lu. Graph homomorphisms with complex values: a dichotomy theorem. *CoRR*, abs/0903.4728, 2009.

[4] J-Y. Cai and P. Lu. Holographic algorithms: from art to science. In *STOC '07: Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 401–410, 2007.

[5] J-Y. Cai, P. Lu, and M. Xia. Holographic algorithms by Fibonacci gates and holographic reductions for hardness. In *FOCS '08: Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 644–653, 2008.

[6] J-Y. Cai, P. Lu, and M. Xia. A computational proof of complexity of some restricted counting problems. In *TAMC '09: Proceedings of Theory and Applications of Models of Computation, 6th Annual Conference*, LNCS 5532, pages 138–149, 2009.

[7] M. Dyer, L.A. Goldberg, and M. Paterson. On counting homomorphisms to directed acyclic graphs. *Journal of the ACM*, 54(6), 2007.

[8] M. Dyer and C. Greenhill. The complexity of counting graph homomorphisms (extended abstract). In *SODA '00: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 246–255, 2000.

[9] L.A. Goldberg, M. Grohe, M. Jerrum, and M. Thurley. A complexity dichotomy for partition functions with mixed signs. *CoRR*, abs/0804.1932, 2008.

[10] P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990.

[11] M. Kowalczyk and J-Y. Cai. Holant Problems for Regular Graphs with Complex Edge Functions. *CoRR*, abs/1001.0464, 2010.

[12] S. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001.

[13] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[14] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science* 8:189–201, 1979.

[15] L. Valiant. Holographic algorithms. *SIAM Journal on Computing*, 37(5):1565–1594, 2008.

[16] L. Valiant. Quantum circuits that can be simulated classically in polynomial time. *SIAM Journal on Computing*, 31(4): 1229–1254, 2002.

[17] M. Xia, P. Zhang, and W. Zhao. Computational complexity of counting problems on 3-regular planar graphs. *Theoretical Computer Science*, 384(1):111–125, 2007.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 537-548
www.stacs-conf.org

IS RAMSEY'S THEOREM ω-AUTOMATIC?

DIETRICH KUSKE

Centre national de la recherche scientifique (CNRS) and
Laboratoire Bordelais de Recherche en Informatique (LaBRI), Bordeaux, France

Abstract. We study the existence of infinite cliques in ω-automatic (hyper-)graphs. It turns out that the situation is much nicer than in general uncountable graphs, but not as nice as for automatic graphs.

More specifically, we show that every uncountable ω-automatic graph contains an uncountable co-context-free clique or anticlique, but not necessarily a context-free (let alone regular) clique or anticlique. We also show that uncountable ω-automatic ternary hypergraphs need not have uncountable cliques or anticliques at all.

Introduction

Every infinite graph has an infinite clique or an infinite anticlique – this is the paradigmatic formulation of Ramsey's theorem [Ram30]. But this theorem is highly non-constructive since there are recursive infinite graphs whose infinite cliques and anticliques are all non-recursive (not even in Σ^0_2, [Joc72], cf. [Gas98, Thm. 4.6]). Recall that a graph is recursive if both its set of nodes and its set of edges can be decided by a Turing machine. Replacing these Turing machines by finite automata, one obtains the more restrictive notion of an *automatic graph*: the set of nodes is a regular set and whether a pair of nodes forms an edge can be decided by a synchronous two-tape automaton (this concept is known since the beginning of automata theory, a systematic study started with [KN95, BG04], see [Rub08] for a recent overview). In this context, the situation is much more favourable: every infinite automatic graph contains an infinite regular clique or an infinite regular anticlique (cf. [Rub08]).

Soon after Ramsey's paper from 1930, authors got interested in a quantitative analysis. For finite graphs, one can ask for the minimal number of nodes that guarantee the existence of a clique or anticlique of some prescribed size. This also makes sense in the infinite: how many nodes are necessary and sufficient to obtain a clique or anticlique of size $\aleph_0$ (Ramsey's theorem tells us: $\aleph_0$) or $\aleph_1$ (here one needs more than $2^{\aleph_0}$ nodes [Sie33, ER56]).

Since automatic graphs contain at most $\aleph_0$ nodes, we need a more general notion for a recursion-theoretic analysis of this situation. For this, we use Blumensath & Grädel's [BG04] ω-automatic graphs: the names of nodes form a regular ω-language and the edge

1998 ACM Subject Classification: F.4.1.
Key words and phrases: Logic in computer science, Automata, Ramsey theory.

These results were obtained when the author was affiliated with the Universität Leipzig.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2483

relation (on names) as well as the relation "these two names denote the same node" can be decided by a synchronous 2-tape Büchi-automaton. In this paper, we answer the question whether these ω-automatic graphs are more like automatic graphs (i.e., large cliques or anticliques with nice properties exist) or like general graphs (large cliques need not exist).

Our answer to this question is a clear "somewhere in between": We show that every ω-automatic graph of size $2^{\aleph_0}$ contains a clique or anticlique of size $2^{\aleph_0}$ (Theorem 3.1) – this is in contrast to the case of arbitrary graphs where such a subgraph need not exist [Sie33]. But in general, there is no regular clique or anticlique (Theorem 3.13) – this is in contrast with the case of automatic graphs where we always find a large regular clique or anticlique. Finally, we also provide an ω-automatic "ternary hypergraph" of size $2^{\aleph_0}$ without any clique or anticlique of size $\aleph_1$, let alone $2^{\aleph_0}$ (Theorem 3.11).

For Theorem 3.1, we re-use the proof from [BKR08] that was originally constructed to deal with infinity quantifiers in ω-automatic structures. The proof of Theorem 3.13 makes use of the "ultimately equal" relation. This relation was also crucial in the separation of injectively from general ω-automatic structures [HKMN08] as well as in the handling of infinity quantifiers in [KL08] and [BKR08]. In the ternary hypergraph from Theorem 3.11, a 3-set $\{x, y, z\}$ of infinite words with $x <_{\text{lex}} y <_{\text{lex}} z$ forms an undirected hyperedge iff the longest common prefix of x and y is shorter than the longest common prefix of y and z.

¿From Theorem 3.1 (i.e., the existence of large cliques or anticliques in ω-automatic graphs), we derive that any ω-automatic partial order of size $2^{\aleph_0}$ contains an antichain of size $2^{\aleph_0}$ or a copy of the real line.

1. Preliminaries

1.1. Ramsey-theory

For a set V and a natural number $k \geq 1$, let $[V]^k$ denote the set of k-element subsets of V. A *(k,ℓ)-partition* is a pair $G = (V, E_1, \dots, E_\ell)$ where V is a set and $(E_1, \dots, E_\ell)$ is a partition of $[V]^k$ into (possibly empty) sets. For $1 \leq i \leq \ell$, a set $W \subseteq V$ is *E_i-homogeneous* if $[W]^k \subseteq E_i$; it is *homogeneous* if it is E_i-homogeneous for some $1 \leq i \leq \ell$. The case $k = \ell = 2$ is special: any $(2,2)$-partition $G = (V, E_1, E_2)$ can be considered as an (undirected loop-free) graph (V, E_1). Homogeneous sets in G are then complete or discrete induced subgraphs of (V, E_1).

Ramsey theory is concerned with the following question: Does every (k,ℓ)-partition $G = (V, E_1, \dots, E_\ell)$ with $|V| = \kappa$ have a homogeneous set of size λ (where κ and λ are cardinal numbers and $k, \ell \geq 2$ are natural numbers). If this is the case, one writes

$$\kappa \to (\lambda)^k_\ell$$

(a notation due to Erdős and Rado [ER56]). This allows to formulate Ramsey's theorem concisely:

Theorem 1.1 (Ramsey [Ram30]). *If $k, \ell \geq 2$, then $\aleph_0 \to (\aleph_0)^k_\ell$.*

In particular, every graph with $\aleph_0$ nodes contains a complete or discrete induced subgraph of the same size. If one wants to find homogeneous sets of size $\aleph_1$, the base set has to be much larger:

Theorem 1.2 (Sierpiński [Sie33]). *If $k, \ell \geq 2$, then $2^{\aleph_0} \not\to (\aleph_1)^k_\ell$ and therefore in particular $2^{\aleph_0} \not\to (2^{\aleph_0})^k_\ell$.*

Erdős and Rado [ER56] proved that partitions of size properly larger than $2^{\aleph_0}$ have homogeneous sets of size $\aleph_1$. For more details on infinite Ramsey theory, see [Jec02, Chapter 9].

1.2. ω-languages

Let Γ be a finite alphabet. With Γ^* we denote the set of all finite words over the alphabet Γ. The set of all nonempty finite words is Γ^+. An *ω-word* over Γ is an infinite ω-sequence $x = a_0a_1a_2\cdots$ with $a_i \in \Gamma$, we set $x[i,j) = a_ia_{i+1}\dots a_{j-1}$ for natural numbers $i \leq j$. In the same spirit, $x[i,\omega)$ denotes the ω-word $a_ia_{i+1}\dots$. The set of all ω-words over Γ is denoted by Γ^ω and $\Gamma^\infty = \Gamma^* \cup \Gamma^\omega$. For a set $V \subseteq \Gamma^+$ of finite words let $V^\omega \subseteq \Gamma^\omega$ be the set of all ω-words of the form $v_0v_1v_2\cdots$ with $v_i \in V$. Two infinite words $x, y \in \Gamma^\omega$ are *ultimately equal*, briefly $x \sim_e y$, if there exists $i \in \mathbb{N}$ with $x[i,\omega) = y[i,\omega)$. By $\leq_{\text{lex}}$, we denote the lexicographic order on the set Σ^ω (with some, implicitly assumed linear order on the letters from Σ) and $\leq_{\text{pref}}$ the prefix order on Σ^∞.

For $\Sigma = \{0,1\}$, the support $\text{supp}(x) \subseteq \mathbb{N}$ is the set of positions of the letter 1 in the word $x \in \Sigma^\omega$.

A (nondeterministic) *Büchi-automaton* M is a tuple $M = (Q, \Gamma, \delta, \iota, F)$ where Q is a finite set of states, $\iota \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta \subseteq Q \times \Gamma \times Q$ is the transition relation. If $\Gamma = \Sigma^n$ for some alphabet Σ, then we speak of an *n-dimensional Büchi-automaton over* Σ. A *run* of M on an ω-word $x = a_0a_1a_2\cdots$ is an ω-word $r = p_0p_1p_2\cdots$ over the set of states Q such that $(p_i, a_i, p_{i+1}) \in \delta$ for all $i \geq 0$. The run r is *successful* if $p_0 = \iota$ and there exists a final state from F that occurs infinitely often in r. The ω-language $L(M) \subseteq \Gamma^\omega$ defined by M is the set of all ω-words that admit a successful run. An ω-language $L \subseteq \Gamma^\omega$ is *regular* if there exists a Büchi-automaton M with $L(M) = L$.

Alternatively, regular ω-languages can be represented algebraically. To this end, one defines *ω-semigroups* to be two-sorted algebras $S = (S_+, S_\omega; \cdot, *, \pi)$ where $\cdot : S_+ \times S_+ \to S_+$ and $* : S_+ \times S_\omega \to S_\omega$ are binary operations and $\pi : (S_+)^\omega \to S_\omega$ is an ω-ary operation such that the following hold:

- $(S_+, \cdot)$ is a semigroup,
- $s * (t * u) = (s \cdot t) * u$,
- $s_0 \cdot \pi((s_i)_{i\geq 1}) = \pi((s_i)_{i\geq 0})$,
- $\pi((s_i^1 \cdot s_i^2 \cdots s_i^{k_i})_{i\geq 0}) = \pi((t_j)_{j\geq 0})$ whenever
$$(t_j)_{j\geq 0} = (s_0^1, s_0^2, \dots, s_0^{k_0}, s_1^1, \dots, s_1^{k_1}, \dots)\ .$$

The ω-semigroup S is *finite* if both, S_+ and S_ω are finite. The free ω-semigroup generated by Γ is
$$\Gamma^\infty = (\Gamma^+, \Gamma^\omega; \cdot, *, \pi)$$
where $u \cdot v$ and $u * x$ are the natural operations of prefixing a word by the finite word u, and $\pi((u_i)_{i\geq 0})$ is the omega-word $u_0u_1u_2\dots$. A homomorphism $h : \Gamma^\infty \to S$ of ω-semigroups maps finite words to elements of S_+ and ω-words to elements of S_ω and commutes with the operations $\cdot$, $*$, and π. The algebraic characterisation of regular ω-languages then reads as follows.

Proposition 1.3. *An ω-language $L \subseteq \Gamma^\omega$ is regular if and only if there exists a finite ω-semigroup S, a set $T \subseteq S_\omega$, and a homomorphism $\eta : \Gamma^\infty \to S$ such that $L = \eta^{-1}(T)$.*

Hence, every Büchi-automaton is "equivalent" to a homomorphism into some finite ω-semigroup together with a distinguished set T (and vice versa).

For ω-words $x_i = a_i^0 a_i^1 a_i^2 \cdots \in \Gamma^\omega$, the *convolution* $x_1 \otimes x_2 \otimes \cdots \otimes x_n \in (\Gamma^n)^\omega$ is defined by

$$(x_1, \dots, x_n)^\otimes = (a_1^0, \dots, x_n^0)\,(a_1^1, \dots, a_n^1)\,(a_1^2, \dots, a_n^2) \cdots \ .$$

An n-ary relation $R \subseteq (\Gamma^\omega)^n$ is called *ω-automatic* if the ω-language $\{(x_1, \dots, x_n)^\otimes \mid (x_1, \dots, x_n) \in R\}$ is regular.

To describe the complexity of ω-languages, we will use language-theoretic terms. Let LANG denote the class of all languages (i.e., sets of finite words over some finite set of symbols) and ωLANG the class of all ω-languages. By REG and ωREG, we denote the regular languages and ω-languages, resp. An ω-language is *context-free* if it can be accepted by a pushdown-automaton with Büchi-acceptance (on states), it is *co-context-free* if its complement is context-free. We denote by ωCF the set of context-free ω-languages and by co-ωCF their complements. An ω-language belongs to LANG* if it is of the form $\bigcup_{1 \le i \le n} U_i V_i^\omega$ with $U_i, V_i \in$ LANG. Then ωREG $\subseteq$ LANG* and ωCF $\subseteq$ LANG* where the sets U_i and V_i are regular and context-free, resp [Sta97]. In between these two classes, we define the class ωerCF of *eventually regular context-free* ω-languages that comprises all sets of the form $\bigcup_{1 \le i \le n} U_i V_i^\omega$ with $U_i \in$ LANG context-free and $V_i \in$ LANG regular. Alternatively, eventually regular context-free ω-languages are the finite unions of ω-languages of the form $C \cdot L$ where C is a context free-language and L a regular ω-language. Let co-ωerCF denote the set of complements of eventually regular context-free ω-languages.

A final, rather peculiar class of ω-languages is $\mathbf{\Lambda}$: it is the class of ω-languages L such that $(\mathbb{R}, \le)$ embeds into $(L, \le_{\text{lex}})$ (the name derives from the notation $\boldsymbol{\lambda}$ for the order type of $(\mathbb{R}, \le)$).

1.3. ω-automatic (k, ℓ)-partitions

An *ω-automatic presentation of a (k, ℓ)-partition* $(V, E_1, \dots, E_\ell)$ is a pair (L, h) consisting of a regular ω-language L and a surjection $h : L \to V$ such that $\{(x_1, x_2, \dots, x_k) \in L^k \mid \{h(x_1), h(x_2), \dots, h(x_k)\} \in E_i\}$ for $1 \le i \le k$ and $R_\approx = \{(x_1, x_2) \in L^2 \mid h(x_1) = h(x_2)\}$ are ω-automatic. An ω-automatic presentation is *injective* if h is a bijection. A (k, ℓ)-partition is *(injectively) ω-automatic* if it has an (injective) ω-automatic presentation. From [BKR08], it follows that an uncountable ω-automatic (k, ℓ)-partition has $2^{\aleph_0}$ elements.

This paper is concerned with the question whether every (injective) ω-automatic presentation (L, h) of a (k, ℓ)-partition admits a "simple" set $H \subseteq L$ such that $h(H)$ has λ elements and is homogeneous. More precisely, let $\mathcal{C}$ be a class of ω-languages, $k, \ell \ge 2$ natural numbers, and κ and λ cardinal numbers. Then we write

$$(\kappa, \omega\mathsf{A}) \to (\lambda, \mathcal{C})^k_\ell$$

if the following partition property holds: for every ω-automatic presentation (L, h) of a (k, ℓ)-partition G of size κ, there exists $H \subseteq L$ in $\mathcal{C}$ such that $h(H)$ is homogeneous in G and of size λ.

$$(\kappa, \omega\mathsf{iA}) \to (\lambda, \mathcal{C})^k_\ell$$

is to be understood similarly where we only consider injective ω-automatic presentations.

Remark 1.4. Let $G = (V, E_1, \dots, E_\ell)$ be some (k, ℓ)-partition with ω-automatic presentation (L, h). Then the partition property above requires that there is a "large" homogeneous set $X \subseteq V$ and an ω-language $H \in \mathcal{C}$ such that $h(H) = X$, in particular, every element of

X has at least one representative in H. Alternatively, one could require that $h^{-1}(X) \subseteq L$ is an ω-language from $\mathcal{C}$. In this paper, we only encounter classes $\mathcal{C}$ of ω-languages such that the following closure property holds: if $H \in \mathcal{C}$ and R is an ω-automatic relation, then also $R(H) = \{y \mid \exists x \in H : (x, y) \in R\} \in \mathcal{C}$. Since $h^{-1}h(H) = R_{\approx}(H)$, all our results also hold for this alternative requirement $h^{-1}(X) \in \mathcal{C}$.

This paper shows

(0) if $k, \ell \geq 2$, then $(\aleph_0, \omega\mathsf{A}) \to (\aleph_0, \omega\mathrm{REG})^k_\ell$, but $(2^{\aleph_0}, \omega\mathsf{A}) \not\to (\aleph_0, \omega\mathrm{REG})^k_\ell$, see Theorem 2.1.

(1) if $\ell \geq 2$, then $(2^{\aleph_0}, \omega\mathsf{A}) \to (2^{\aleph_0}, \text{co-}\omega\mathrm{erCF})^2_\ell$, see Theorem 3.1.

(2) if $k \geq 3$, $\ell \geq 2$, and $\lambda > \aleph_0$, then $(2^{\aleph_0}, \omega\mathsf{iA}) \not\to (\lambda, \omega\mathrm{LANG})^k_\ell$, see Theorem 3.11.

(3) if $k, \ell \geq 2$ and $\lambda > \aleph_0$, then $(2^{\aleph_0}, \omega\mathsf{iA}) \not\to (\lambda, \omega\mathrm{CF})^k_\ell$, see Theorem 3.13.

Here, the first part of (0) is a strengthening of Ramsey's theorem since the infinite homogeneous set is regular. The second part might look surprising since larger (k, ℓ)-partitions should have larger homogeneous sets – but not necessarily regular ones! In contrast to Sierpiński's result, (1) shows that ω-automatic $(2, \ell)$-partitions have a larger degree of homogeneity than arbitrary $(2, \ell)$-partitions. Even more, the complexity of the homogeneous set can be bound in language-theoretic terms (there is always a homogeneous set that is the complement of an eventually regular context-free ω-language). Statement (2) is an analogue of Sierpiński's Theorem 1.2 showing that (injective) ω-automatic (k, ℓ)-partitions are as in-homogeneous as arbitrary (k, ℓ)-partitions provided $k \geq 3$. The complexity bound from (1) is shown to be optimal by (3) proving that one cannot always find context-free homogeneous sets. Hence, despite the existence of large homogeneous sets for $k = 2$, for some ω-automatic presentations, they are bound to have a certain (low) level of complexity that is higher than the regular ω-languages.

2. Countably infinite homogeneous sets

Let $k, \ell \geq 2$ be arbitrary. Then, from Ramsey's theorem, we obtain immediately $(\aleph_0, \omega\mathsf{A}) \to (\aleph_0, \omega\mathrm{LANG})^k_\ell$ and $(2^{\aleph_0}, \omega\mathsf{A}) \to (\aleph_0, \omega\mathrm{LANG})^k_\ell$, i.e., all infinite ω-automatic (k, ℓ)-partitions have homogeneous sets of size $\aleph_0$. In this section, we ask whether such homogeneous sets can always be chosen regular:

Theorem 2.1. *Let $k, \ell \geq 2$. Then*

(a) $(\aleph_0, \omega\mathsf{A}) \to (\aleph_0, \omega\mathrm{REG})^k_\ell$.

(b) $(2^{\aleph_0}, \omega\mathsf{iA}) \to (\aleph_0, \omega\mathrm{REG})^k_\ell$.

(c) $(2^{\aleph_0}, \omega\mathsf{A}) \not\to (\aleph_0, \mathrm{LANG}^*)^k_\ell$, *and therefore in particular* $(2^{\aleph_0}, \omega\mathsf{A}) \not\to (\aleph_0, \omega\mathrm{CF})^k_\ell$ *and* $(2^{\aleph_0}, \omega\mathsf{A}) \not\to (\aleph_0, \omega\mathrm{REG})^k_\ell$.

Proof. Let (L, h) be an ω-automatic presentation of some (k, ℓ)-partition $G = (V, E_1, \ldots, E_\ell)$ with $|V| = \aleph_0$. By [BKR08], there exists $L' \subseteq L$ regular such that (L', h) is an injective ω-automatic presentation of G. From a Büchi-automaton for L', one can compute a finite automaton accepting some language K such that (K, h') is an injective automatic presentation of G [Blu99]. Hence, by [Rub08], there exists a regular set $H' \subseteq K$ such that $h'(H')$ is homogeneous in G and countably infinite. From this set, one obtains a regular ω-language $H \subseteq L' \subseteq L$ with $h(H) = h'(H')$, i.e., $h(H)$ is a homogeneous set of size $\aleph_0$. This proves (a).

To prove (b), let (L, h) be an injective ω-automatic presentation of some (k, ℓ)-partition $G = (V, E_1, \ldots, E_\ell)$ of size $2^{\aleph_0}$. Then there exists a regular ω-language $L' \subseteq L$ with

$|L'| = \aleph_0$. Consider the sub-partition $G' = (h(L'), E'_1, \dots, E'_\ell)$ with $E'_i = E_i \cap [h(L')]^k$. This (k,ℓ)-partition has as ω-automatic presentation the pair (L', h). Then, by (a), there exists $L'' \subseteq L'$ regular and infinite such that $h(L'')$ is homogeneous in G' and therefore in G. Since h is injective, this implies $|h(L')| = |L'| = \aleph_0$.

Finally, we show (c) by a counterexample. Let $L = \{0,1\}^\omega$, $V = L/{\sim_e}$, and $h : L \to V$ the canonical mapping. Furthermore, set $E_1 = [L]^k$. Then $G = (V, E_1, \emptyset, \dots, \emptyset)$ is a (k,ℓ)-partition with ω-automatic presentation (L, h).

Now let $H = \bigcup_{1 \le i \le n} U_i V_i^\omega \subseteq L$ for some non-empty languages $U_i, V_i \subseteq \{0,1\}^+$ such that $h(H)$ is homogeneous and infinite.

If $|V_i^\omega| = 1$, then $U_i V_i^\omega/{\sim_e}$ is finite. Since $h(H)$ is infinite, there exists $1 \le i \le n$ with $|V_i^\omega| > 1$ implying the existence of words $v, w \in V_i^+$ such that $|v| = |w|$ and $v \ne w$. For $u \in U_i$, the set $u\{v,w\}^\omega \subseteq H$ has $2^{\aleph_0}$ equivalence classes wrt. $\sim_e$. Hence $|h(H)| = 2^{\aleph_0}$. ∎

3. Uncountable homogeneous sets

3.1. A Ramsey theorem for ω-automatic $(2,\ell)$-partitions

The main result of this section is the following theorem that follows immediately from Prop. 3.7 and Lemma 3.5.

Theorem 3.1. *For all $\ell \ge 2$, we have $(2^{\aleph_0}, \omega\mathsf{A}) \to (2^{\aleph_0}, \text{co-}\omega\text{erCF} \cap \mathbf{\Lambda})^2_\ell$.*

3.1.1. *The proof.* The proof of this theorem will construct a language from co-ωerCF that describes a homogeneous set. This language is closely related to the following language

$$N = 1\{0,1\}^\omega \cap \bigcap_{n \ge 0} \{0,1\}^n (0\{0,1\}^n 00 \cup 10^n\{01,10\})\{0,1\}^\omega \ ,$$

i.e., an ω-word x belongs to N iff it starts with 1 and, for every $n \ge 0$, we have $x[n, 2n+3) \in 0\{0,1\}^*00 \cup 10^*01 \cup 10^*10$. We first list some useful properties of this language N:

Lemma 3.2. *The ω-language N is contained in $(1^+0^+)^\omega$, belongs to* co-ωerCF $\cap\, \mathbf{\Lambda}$*, and* $\mathrm{supp}(x) \cap \mathrm{supp}(y)$ *is finite for any $x, y \in N$ distinct.*

Proof. Let $b_i \in \{0,1\}$ for all $i \ge 0$ and suppose the word $x = b_0 b_1 \dots$ belongs to N. Then $b_0 = 1$, hence the word x contains at least one occurrence of 1. Note that, whenever $b_n = 1$, then $\{b_{2n+1}, b_{2n+2}\} = \{0,1\}$, hence x contains infinitely many occurrences of 1 and therefore infinitely many occurrences of 0, i.e., $N \subseteq (1^+0^+)^\omega$.

Note that the complement of N equals

$$0\{0,1\}^\omega \cup \bigcup_{n \ge 0} \Big(\{0,1\}^n (0\{0,1\}^n\{01,10,11\} \cup 1\{0,1\}^n\{00,11\})\{0,1\}^\omega\Big)$$

$$= \left[0 \cup \bigcup_{n \ge 0} \{0,1\}^n (0\{0,1\}^n\{01,10,11\} \cup 1\{0,1\}^n\{00,11\})\right] \{0,1\}^\omega \ .$$

Since the expression in square brackets denotes a context-free language, $\{0,1\}^\omega \setminus N$ is an eventually regular context-free ω-language.

Note that a word $10^{n_0}10^{n_1}10^{n_2}\dots$ belongs to N iff, for all $k \geq 0$, we have $0 \leq n_k - |10^{n_0}10^{n_1}\dots 10^{n_{k-1}}| \leq 1$. Hence, when building a word from N, we have two choices for any n_k, say n_k^0 and n_k^1 with $n_k^0 < n_k^1$. But then $a_0a_1a_2\dots \mapsto 10^{n_0^{a_0}}10^{n_1^{a_1}}10^{n_2^{a_2}}\dots$ defines an order embedding $(\{0,1\}^\omega, \leq_{\text{lex}}) \hookrightarrow (N, \leq_{\text{lex}})$. Since $(\mathbb{R}, \leq) \hookrightarrow (\{0,1\}^\omega, \leq_{\text{lex}})$, we get $N \in \mathbf{\Lambda}$.

Now let $x, y \in N$ with $\text{supp}(x) \cap \text{supp}(y)$ infinite. Then there are arbitrarily long finite words u and v of equal length such that $u1$ and $v1$ are prefixes of x and y, resp. Since $u1$ is a prefix of $x \in N$, it is of the form $u1 = u'10^{|u'|}1$ (if $|u|$ is even) or $u1 = u'10^{|u'|}01$ (if $|u|$ is odd) and analogously for v. Inductively, one obtains $u' = v'$ and therefore $u = v$. Since u and v are arbitrarily long, we showed $x = y$. ∎

Lemma 3.3. *Let $\sim$ and $\approx$ be two equivalence relations on some set L such that any equivalence class $[x]_\sim$ of $\sim$ is countable and $\approx$ has $2^{\aleph_0}$ equivalence classes. Then there are elements $(x_\alpha)_{\alpha<2^{\aleph_0}}$ of L such that $[x_\alpha]_{\sim_e} \cap [x_\beta]_\approx = \emptyset$ for all $\alpha < \beta$.*

Proof. We construct the sequence $(x_\alpha)_{\alpha<2^{\aleph_0}}$ by ordinal induction. So assume we have elements $(x_\alpha)_{\alpha<\kappa}$ for some ordinal $\kappa < 2^{\aleph_0}$ with $[x_\alpha]_\sim \cap [x_\beta]_\approx = \emptyset$ for all $\alpha < \beta < \kappa$.

Suppose $\bigcup_{\alpha<\kappa}[x_\alpha]_\sim \cap [x]_\approx \neq \emptyset$ for all $x \in L$. For $x, y \in L$ with $x \not\approx y$, we have $(\bigcup_{\alpha<\kappa}[x_\alpha]_\sim \cap [x]_\approx) \cap (\bigcup_{\alpha<\kappa}[x_\alpha]_\sim \cap [y]_\approx) \subseteq [x]_\approx \cap [y]_\approx = \emptyset$. Since $\bigcup_{\alpha<\kappa}[x_\alpha]_\sim$ has $\kappa \cdot \aleph_0 \leq \max(\kappa, \aleph_0) < 2^{\aleph_0}$ elements, we obtain $|L| < 2^{\aleph_0}$, contradicting $|L| \geq |L/{\approx}| = 2^{\aleph_0}$. Hence there exists an element $x_\kappa \in L$ with $[x_\alpha]_\sim \cap [x_\kappa]_\approx = \emptyset$ for all $\alpha < \kappa$. ∎

Definition 3.4. Let u, v, and w be nonempty words with $|v| = |w|$ and $v \neq w$. Define an ω-semigroup homomorphism $h : \{0,1\}^\infty \to \Sigma^\infty$ by $h(0) = v$ and $h(1) = w$ and set

$$H_{u,v,w} = u \cdot h(N)$$

where N is the set from Lemma 3.2.

Lemma 3.5. *Let u, v, and w be as in the previous definition. Then $H_{u,v,w} \in$ co-ωerCF$\cap\mathbf{\Lambda}$.*

Proof. Assume $v <_{\text{lex}} w$. Then the mapping $\chi : \{0,1\}^\omega \to \Sigma^\omega : x \mapsto uh(x)$ (where h is the homomorphism from the above definition) embeds $(N, \leq_{\text{lex}})$ (and hence $(\mathbb{R}, \leq)$) into $(H_{u,v,w}, \leq_{\text{lex}})$. If $w <_{\text{lex}} v$, then $(\mathbb{R}, \leq) \cong (\mathbb{R}, \geq) \hookrightarrow (N, \geq_{\text{lex}}) \hookrightarrow (H_{\alpha,\beta,\gamma}, \leq_{\text{lex}})$. This proves that $H_{u,v,w}$ belongs to $\mathbf{\Lambda}$.

Since $v \neq w$, the mapping χ is injective. Hence

$$\Sigma^\omega \setminus H_{\alpha,\beta,\gamma} = \Sigma^\omega \setminus \chi(N) = \Sigma^\omega \setminus \chi(\{0,1\}^\omega) \cup \chi(\{0,1\}^\omega \setminus N)\ .$$

Since χ can be realized by a generalized sequential machine with Büchi-acceptance, $\chi(\{0,1\}^\omega)$ is regular and $\chi(\{0,1\}^\omega \setminus N)$ (as the image of an eventually regular context-free ω-language) is eventually regular context-free. Hence $\Sigma^\omega \setminus H_{u,v,w}$ is eventually regular context-free. ∎

Proposition 3.6. *Let $G = (L, E_0, E_1, \dots, E_\ell)$ be some $(2, 1+\ell)$-partition with injective ω-automatic presentation (L, id) such that $\{(x,y) \mid \{x,y\} \in E_0\} \cup \{(x,x) \mid x \in L\}$ is an equivalence relation on L (denoted $\approx$) with $2^{\aleph_0}$ equivalence classes. Then there exist nonempty words u, v, and w with v and w distinct, but of the same length, such that $H_{u,v,w}$ is i-homogeneous for some $1 \leq i \leq \ell$.*

Proof. There are finite ω-semigroups S and T and homomorphisms $\gamma : \Sigma^\infty \to S$ and $\delta : (\Sigma \times \Sigma)^\infty \to T$ such that

(a) $x \in L$, $y \in \Sigma^\omega$, and $\gamma(x) = \gamma(y)$ imply $y \in L$ and

(b) $x, x', y, y' \in L$, $\{h(x), h(x')\} \in E_i$, and $\delta(x, x') = \delta(y, y')$ imply $\{h(y), h(y')\} \in E_i$ (for all $0 \leq i \leq \ell$).

By Lemma 3.3, there are words $(x_\alpha)_{\alpha<2^{\aleph_0}}$ in L such that $[x_\alpha]_{\sim_e} \cap [x_\beta]_\approx = \emptyset$ for all $\alpha < \beta$.

In the following, we only need the words $x_0, x_1, \ldots, x_C$ with $C = |S| \cdot |T|$. Then [BKR08, Sections 3.1-3.3][1] first constructs two ω-words y_1 and y_2 and an infinite sequence $1 \leq g_1 < g_2 < \ldots$ of natural numbers such that in particular $y_1[g_1, g_2) <_{\text{lex}} y_2[g_1, g_2)$. Set $u = y_2[0, g_1)$, $v = y_1[g_1, g_2)$, and $w = y_2[g_1, g_2)$. In the following, let $h : \{0,1\}^\infty \to \Sigma^\infty$ be the homomorphism from Def. 3.4 and set $\chi(x) = uh(x)$ for $x \in \{0,1\}^*$. As in [BKR08], one can then show that all the words from $H_{u,v,w}$ belong to the ω-language L. In the following, set $x_{\circ\bullet} = \chi((01)^\omega)$ and $x_{\bullet\circ} = \chi((10)^\omega)$. Then obvious alterations in the proofs by Bárány et al. show:

(1) [BKR08, Lemma 3.4][2] If $x, y \in \{0,1\}^\omega$ with $\text{supp}(x) \setminus \text{supp}(y)$ and $\text{supp}(y) \setminus \text{supp}(x)$ infinite, then
$$\{\delta(\chi(x), \chi(y)), \delta(\chi(y), \chi(x))\} = \{\delta(x_{\bullet\circ}, x_{\circ\bullet}), \delta(x_{\circ\bullet}, x_{\bullet\circ})\} \ .$$

(2) [BKR08, Lemma 3.5] $x_{\bullet\circ} \not\approx x_{\circ\bullet}$.

There exists $0 \leq i \leq \ell$ with $\{x_{\bullet\circ}, x_{\circ\bullet}\} \in E_i$. Then (2) implies $i > 0$.

Let $x, y \in N$ be distinct. Then $\text{supp}(x) \cap \text{supp}(y)$ is finite by Lemma 3.2. Since, on the other hand, $\text{supp}(x)$ and $\text{supp}(y)$ are both infinite, the two differences $\text{supp}(x) \setminus \text{supp}(y)$ and $\text{supp}(y) \setminus \text{supp}(x)$ are infinite. Hence we obtain $\delta(\chi(x), \chi(y)) \in \{\delta(x_{\bullet\circ}, x_{\circ\bullet}), \delta(x_{\circ\bullet}, x_{\bullet\circ})\}$ from (1). Hence (b) implies $\{\chi(x), \chi(y)\} \in E_i$, i.e., $H_{u,v,w}$ is E_i-homogeneous.

Since $H_{u,v,w} \in \text{co-}\omega\text{erCF} \cap \mathbf{\Lambda}$ by Lemma 3.5, the result follows. ∎

Proposition 3.7. *Let $G = (V, E'_1, \ldots, E'_\ell)$ be some $(2,\ell)$-partition with automatic presentation (L, h). Then there exist $u, v, w \in \Sigma^+$ with v and w distinct of equal length such that $h(H_{u,v,w})$ is homogeneous and of size $2^{\aleph_0}$.*

Proof. To apply Prop. 3.6, consider the following $(2, 1+\ell)$-partition $G = (L, E_0, \ldots, E_\ell)$:

- The underlying set is the ω-language L,
- E_0 comprises all sets $\{x, y\}$ with $h(x) = h(y)$ and $x \neq y$, and
- E_i (for $1 \leq i \leq \ell$) comprises all sets $\{x, y\}$ with $\{h(x), h(y)\} \in E'_i$.

Then (L, id) is an injective ω-automatic presentation of the $(2, 1+\ell)$-partition G. By Prop. 3.6, there exists $1 \leq i \leq \ell$ and words u, v and w such that $H_{u,v,w}$ is i-homogeneous in G. Since $(E_0, \ldots, E_\ell)$ is a partition of $[L]^2$, we have $\{x, y\} \notin E_0$ (and therefore $h(x) \neq h(y)$) for all $x, y \in H_{u,v,w}$ distinct. Hence h is injective on $H_{u,v,w}$. Furthermore $[H_{u,v,w}]^2 \subseteq E_i$ implies $[h(H_{u,v,w})]^2 \subseteq E'_i$. Hence $h(H_{u,v,w})$ is an i-homogeneous set in G' of size $2^{\aleph_0}$. ∎

This finishes the proof of Theorem 3.1.

3.1.2. *Effectiveness.* Note that the proof above is non-constructive at several points: Lemma 3.3 is not constructive and the proof proper uses Ramsey's theorem [BKR08, page 390] and makes a Ramseyan factorisation coarser [BKR08, begin of section 3.2]. We now show that nevertheless the words u, v, and w can be computed. By Prop. 3.7, it suffices to decide for a given triple (u, v, w) whether $h(H_{u,v,w})$ is i-homogeneous for some fixed $1 \leq i \leq \ell$.

To be more precise, let $(V, E_1, \ldots, E_\ell)$ be some $(2,\ell)$-partition with ω-automatic presentation (L, h). Furthermore, let $u, v, w \in \Sigma^+$ with $v \neq w$ of the same length and write H

[1] The authors of [BKR08] require $[x_i]_{\sim_e} \cap [x_j]_\approx = \emptyset$ for all $0 \leq i, j \leq C$ distinct, but they use it only for $i < j$. Hence we can apply their result here.

[2] The authors of [BKR08] only require one of the two differences to be infinite, but the proof uses that they both are infinite.

for $H_{u,v,w}$. We have to decide whether $H \subseteq L$ and $H \otimes H \subseteq L_i \cup L_=$. Note that $H \subseteq L$ iff $L \cap \Sigma^\omega \setminus H = \emptyset$. But $\Sigma^\omega \setminus H$ is context-free, so the intersection is context-free. Hence the emptiness of the intersection can be decided.

Towards a decision of the second requirement, note that

$$(\Sigma \times \Sigma)^\omega \setminus (H \otimes H) = (\Sigma^\omega \setminus H \otimes \Sigma^\omega) \cup (\Sigma^\omega \cup \Sigma^\omega \setminus H)$$

is the union of two context-free ω-languages and therefore context-free itself. Since $L_i \cup L_=$ is regular, the intersection $(L_i \cup L_=) \cap (\Sigma \times \Sigma)^\omega \setminus (H \otimes H)$ is context-free implying that its emptiness is decidable. But this emptiness is equivalent to $H \otimes H \subseteq L_1 \cup L_=$.

3.1.3. *ω-automatic partial orders.* ¿From Theorem 3.1, we now derive a necessary condition for a partial order of size $2^{\aleph_0}$ to be ω-automatic. A partial order $(V, \sqsubseteq)$ is ω-automatic iff there exists a regular ω-language L and a surjection $h : L \to V$ such that the relations $R_= = \{(x, y) \in L^2 \mid h(x) = h(y)\}$ and $R_\sqsubseteq = \{(x, y) \in L^2 \mid h(x) \sqsubseteq h(y)\}$ are ω-automatic.

Corollary 3.8 ([BKR08][3])**.** *If $(V, \sqsubseteq)$ is an ω-automatic partial order with $|V| \geq \aleph_1$, then $(\mathbb{R}, \leq)$ or an antichain of size $2^{\aleph_0}$ embeds into $(V, \sqsubseteq)$.*

Proof. Let $(V, \sqsubseteq)$ be a partial order, $L \subseteq \Sigma^\omega$ a regular ω-language and $h : L \to V$ a surjection such that $R_=$ and $R_\sqsubseteq$ are ω-automatic. Define an injective ω-automatic $(2, 4)$-partition $G = (L, E_0, E_1, E_2, E_3)$:

- E_0 comprises all pairs $\{x, y\} \in [L]^2$ with $h(x) = h(y)$,
- E_1 comprises all pairs $\{x, y\} \in [L]^2$ with $h(x) \sqsubset h(y)$ and $x <_{\text{lex}} y$,
- E_2 comprises all pairs $\{x, y\} \in [L]^2$ with $h(x) \sqsupset h(y)$ and $x <_{\text{lex}} y$, and
- $E_3 = [L]^2 \setminus (E_0 \cup E_1 \cup E_2)$ comprises all pairs $\{x, y\} \in [L]^2$ such that $h(x)$ and $h(y)$ are incomparable.

From $|L| \geq |V| > \aleph_0$, we obtain $|L| = 2^{\aleph_0}$. Hence, by Prop. 3.6, there exists $H \subseteq L$ 1-, 2- or 3-homogeneous with $(\mathbb{R}, \leq) \hookrightarrow (H, \leq_{\text{lex}})$. Since $[H]^2 \subseteq E_1 \cup E_2 \cup E_3$ and since G is a partition of L, the mapping h acts injectively on H. If $[H]^2 \subseteq E_1$ (the case $[H]^2 \subseteq E_2$ is symmetrical) then $(\mathbb{R}, \leq) \hookrightarrow (H, \leq_{\text{lex}}) \cong (h(H), \sqsubseteq)$. If $[H]^2 \subseteq E_3$, then $h(H)$ is an antichain of size $2^{\aleph_0}$. ■

A linear order $(L, \sqsubseteq)$ is *scattered* if $(\mathbb{Q}, \leq)$ cannot be embedded into $(L, \sqsubseteq)$. Automatic partial orders are defined similarly to ω-automatic partial orders with the help of finite automata instead of Büchi-automata.

Corollary 3.9 ([BKR08][3])**.** *Any scattered ω-automatic linear order $(V, \sqsubseteq)$ is countable. Hence,*

- *a scattered linear order is ω-automatic if and only if it is automatic, and*
- *an ordinal α is ω-automatic if and only if $\alpha < \omega^\omega$.*

Proof. If $(V, \sqsubseteq)$ is not countable, then it embeds $(\mathbb{R}, \leq)$ by the previous corollary and therefore in particular $(\mathbb{Q}, \leq)$. The remaining two claims follow immediately from [BKR08] ("countable ω-automatic structures are automatic") and [Del04] ("an ordinal is automatic iff it is properly smaller than ω^ω"), resp. ■

[3] As pointed out by two referees, the paragraph before Sect. 4.1 in [BKR08] already hints at this result, although in a rather implicit way.

Contrast Theorem 3.1 with Theorem 1.2: any uncountable ω-automatic (k,ℓ)-partition contains an uncountable homogeneous set of size $2^{\aleph_0}$. But we were able to prove this for $k=2$, only. One would also wish the homogeneous set to be regular and not just from co-ωerCF. We now prove that these two shortcomings are unavoidable: Theorem 3.1 does not hold for $k=3$ nor is there always an ω-regular homogeneous set. These negative results hold even for injective presentations.

3.2. A Sierpiński theorem for ω-automatic (k,ℓ)-partitions with $k \geq 3$

We first concentrate on the question whether some form of Theorem 3.1 holds for $k \geq 3$. The following lemma gives the central counterexample for $k=3$ and $\ell=2$, the below theorem then derives the general result.

Lemma 3.10. $(2^{\aleph_0}, \omega\mathsf{iA}) \not\to (\aleph_1, \omega\mathrm{LANG})^3_2$.

Proof. Let $\Sigma=\{0,1\}$, $V=L=\{0,1\}^\omega$. Furthermore, for $H \subseteq L$, we write $\bigwedge H \in \Sigma^\infty$ for the longest common prefix of all ω-words in H, $\bigwedge\{x,y\}$ is also written $x \wedge y$. Then let E_1 consist of all 3-sets $\{x,y,z\} \in [L]^3$ with $x <_{\text{lex}} y <_{\text{lex}} z$ and $x \wedge y <_{\text{pref}} y \wedge z$; E_2 is the complement of E_1. This finishes the construction of the $(3,2)$-partition (V,E_1,E_2) of size $2^{\aleph_0}$ with injective ω-automatic presentation (L,id).

Note that 1^*0^ω is a countable E_1-homogeneous set and that 0^*1^ω is a countable E_2-homogeneous set. But there is no uncountable homogeneous set: First suppose $H \subseteq L$ is infinite and $x \wedge y <_{\text{pref}} y \wedge z$ for all $x <_{\text{lex}} y <_{\text{lex}} z$ from H. Let $u \in \Sigma^*$ such that $H \cap u0\Sigma^\omega$ and $H \cap u1\Sigma^\omega$ are both nonempty and let $x,y \in H \cap u0\Sigma^\omega$ with $x \leq_{\text{lex}} y$ and $z \in H \cap u1\Sigma^\omega$. Then $x \wedge y >_{\text{pref}} u = y \wedge z$ and therefore $x=y$ (for otherwise, we would have $x <_{\text{lex}} y <_{\text{lex}} z$ in H with $x \wedge y >_{\text{pref}} y \wedge z$). Hence we showed $|H \cap u0\Sigma^\omega|=1$. Let $u_0=\bigwedge H$ and $H_1 = H \cap u_01\Sigma^\omega$. Since $H \cap u_00\Sigma^\omega$ is finite, the set H_1 is infinite. We proceed by induction: $u_n=\bigwedge H_n$ and $H_{n+1}=H_n \cap u_n1\Sigma^\omega$ satisfying $|H_n \cap u_n0\Sigma^\omega|=1$. Then $u_0 <_{\text{pref}} u_01 \leq_{\text{pref}} u_1 <_{\text{pref}} u_11 \leq_{\text{pref}} u_2 \cdots$ with

$$H = \bigcup_{n\geq 0}(H \cap u_n0\Sigma^\omega) \cup \bigcap_{n \geq 0}(H \cap u_n1\Sigma^\omega)\ .$$

Then any of the sets $H \cap u_n0\Sigma^\omega = H_n \cap u_n0\Sigma^\omega$ and $\bigcap(H \cap u_n1\Sigma^\omega)$ is a singleton, proving that H is countable. Thus, there cannot be an uncountable E_1-homogeneous set.

So let $H \subseteq L$ be infinite with $x \wedge y \geq_{\text{pref}} y \wedge z$ for all $x <_{\text{lex}} y <_{\text{lex}} z$. Since we have only two letters, we get $x \wedge y >_{\text{pref}} y \wedge z$ for all $x <_{\text{lex}} y <_{\text{lex}} z$ which allows to argue symmetrically to the above. Thus, indeed, there is no uncountable homogeneous set in L. ■

Theorem 3.11. *For all $k \geq 3$, $\ell \geq 2$, and $\lambda > \aleph_0$, we have* $(2^{\aleph_0}, \omega\mathsf{iA}) \not\to (\lambda, \omega\mathrm{LANG})^k_\ell$.

Proof. Let G be the $(3,2)$-partition from Lemma 3.10 that does not have homogeneous sets of size λ and let (L,id) be an injective ω-automatic presentation of $G=(V,E_1,E_2)$ (in particular, $V=L$).

For a set $X \in [L]^k$, let $X_1 <_{\text{lex}} X_2 <_{\text{lex}} X_3$ be the three lexicographically least elements of X. Then set $G'=(V,E'_1,E'_2,\ldots,E'_\ell)$ with

$$\begin{aligned} E'_1 &= \{X \in [V]^k \mid \{X_1,X_2,X_3\} \in E_1\},\\ E'_2 &= \{X \in [V]^k \mid \{X_1,X_2,X_3\} \in E_2\}, \text{ and}\\ E'_i &= \emptyset \text{ for } 3 \leq i \leq \ell\ . \end{aligned}$$

Then (L, id) is an injective ω-automatic presentation of G'. Now suppose $H' \subseteq L$ is homogeneous in G' and of size λ. Then there exists $H \subseteq H'$ of size λ such that for any words $x_1 <_{\mathrm{lex}} x_2 <_{\mathrm{lex}} x_3$ from H, there exists $X \subseteq H'$ with $X_i = x_i$ for $1 \leq i \leq 3$ (if necessary, throw away some lexicographically largest elements of H'). Hence H is homogeneous in G, contradicting Lemma 3.10. ■

3.3. Complexity of homogeneous sets in ω-automatic $(2, \ell)$-partitions

Having shown that $k = 2$ is a central assumption in Theorem 3.1, we now turn to the question whether homogeneous sets of lower complexity can be found.

Construction. Let $V = L$ denote the regular ω-language $(1^+0^+)^\omega$. Furthermore, $E_1 \subseteq [L]^2$ comprises all 2-sets $\{x, y\} \subseteq L$ such that $\mathrm{supp}(x) \cap \mathrm{supp}(y)$ is finite or $x \sim_e y$. The set E_2 is the complement of E_1 in $[L]^2$. This completes the construction of the $(2, 2)$-partition $G = (L, E_1, E_2)$. Note that (L, id_L) is an injective ω-automatic presentation of G.

By Theorem 3.1, G has an E_1- or an E_2-homogeneous set of size $2^{\aleph_0}$. We convince ourselves that G has large homogeneous sets of both types. By Lemma 3.2, there is an ω-language $N \subseteq (1^+0^+)^\omega$ of size $2^{\aleph_0}$ such that the supports of any two words from N have finite intersection. Hence $[N]^2 \subseteq E_1$ and N has size $2^{\aleph_0}$. But there is also an E_2-homogeneous set L_2 of size $2^{\aleph_0}$: Note that the words from N are mutually non-$\sim_e$-equivalent and let L_2 denote the set of all words $1a_1 1a_2 1a_3 \dots$ for $a_1 a_2 a_3 \dots \in N$. Then for any $x, y \in L_2$ distinct, we have $2\mathbb{N} \subseteq \mathrm{supp}(x) \cap \mathrm{supp}(y)$ and $x \not\sim_e y$, i.e., $\{x, y\} \in E_2$.

Lemma 3.12. *Let $H \in \mathrm{LANG}^*$ have size $\lambda > \aleph_0$. Then H is not homogeneous in G.*

Proof. By definition of LANG^*, there are languages $U_i, V_i \in \mathrm{LANG}$ with $H = \bigcup_{1 \leq i \leq n} U_i V_i^\omega$.

Since H is infinite, there are $1 \leq i \leq n$ and $x, y \in U_i V_i^\omega$ distinct with $x \sim_e y$ and therefore $\{x, y\} \in E_1$.

Since $|H| > \aleph_0$, there is $1 \leq i \leq n$ with $|U_i V_i^\omega| > \aleph_0$; we set $U = U_i$ and $V = V_i$. From $|U| \leq \aleph_0$, we obtain $|V^\omega| > \aleph_0$. Hence there are $v_1, v_2 \in V^+$ distinct with $|v_1| = |v_2|$. Since $uv_1^\omega \in H$ and each element of H contains infinitely many occurrences of 1, the word v_1 belongs to $\{0, 1\}^* 10^*$. Let $u \in U$ be arbitrary (such a word exists since $UV^\omega \neq \emptyset$) and consider the ω-words $x' = u(v_1 v_2)^\omega$ and $y' = u(v_1 v_1)^\omega$ from $UV^\omega \subseteq H$. Then $x' \not\sim_e y'$ since $v_1 \neq v_2$ and $|v_1| = |v_2|$. At the same time, $\mathrm{supp}(x') \cap \mathrm{supp}(y')$ is infinite since v_1 contains an occurrence of 1. Hence $\{x', y'\} \in E_2$.

Thus, we found ω-words $x, y, x', y' \in H$ with $\{x, y\} \in E_1$ and $\{x', y'\} \notin E_1$ proving that H is not homogeneous. ■

Thus, we found a $(2, 2)$-partition $G = (V, E_1, E_2)$ with $2^{\aleph_0}$ elements and an injective ω-automatic presentation (L, h) such that

(1) G has sets L_1 and L_2 in co-ωerCF of size $2^{\aleph_0}$ with $[L_i]^2 \subseteq E_i$ for $1 \leq i \leq 2$.
(2) There is no ω-language $H \in \mathrm{LANG}^*$ with $H \subseteq L$ such that $h(H)$ is homogeneous of size $2^{\aleph_0}$.

Since all context-free ω-languages belong to LANG^*, the following theorem follows the same way that Lemma 3.10 implied Theorem 3.11.

Theorem 3.13. *For all $k, \ell \geq 2$ and $\lambda > \aleph_0$, we have $(2^{\aleph_0}, \omega\mathsf{iA}) \not\to (\lambda, \omega\mathrm{CF})^k_\ell$ and $(2^{\aleph_0}, \omega\mathsf{iA}) \not\to (\lambda, \omega\mathrm{REG})^k_\ell$.*

This result can be understood as another Sierpiński theorem for ω-automatic (k,ℓ)-partitions. This time, it holds for all $k \geq 2$ (not only for $k \geq 3$ as Theorem 3.11). The price to be paid for this is the restriction of homogeneous sets to “simple” ones. In particular the non-existence f regular homogeneous sets provides a Sierpiński theorem in the spirit of automatic structures.

Open questions

Our positive result Theorem 3.1 guarantees the existence of some clique or anticlique of size $2^{\aleph_0}$ (and such a clique or anticlique can even be constructed). But the following situation is conceivable: the ω-automatic graph contains large cliques without containing large cliques that can be described by a language from co-ωerCF. In particular, it is not clear whether the existence of a large clique is decidable.

A related question concerns Ramsey quantifiers. Rubin [Rub08] has shown that the set of nodes of an automatic graph whose neighbors contain an infinite anticlique is regular (his result is much more general, but this formulation suffices for our purpose). It is not clear whether this also holds for ω-automatic graphs. A positive answer to this second question (assuming that it is effective) would entail an affirmative answer to the decidability question above.

References

[BG04] A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Computing Systems*, 37(6):641–674, 2004.

[BKR08] V. Bárány, Ł. Kaiser, and S. Rubin. Cardinality and counting quantifiers on omega-automatic structures. In *STACS'08*, pages 385–396. IFIB Schloss Dagstuhl, 2008.

[Blu99] A. Blumensath. Automatic structures. Technical report, RWTH Aachen, 1999.

[Del04] Ch. Delhommé. Automaticité des ordinaux et des graphes homogènes. *C. R. Acad. Sci. Paris, Ser. I*, 339:5–10, 2004.

[ER56] P. Erdős and R. Rado. A partition calculus in set theory. *Bull. AMS*, 62:427–489, 1956.

[Gas98] W. Gasarch. A survey of recursive combinatorics. In *Handbook of recursive mathematics vol. 2*, volume 139 of *Stud. Logic Found. Math.*, pages 1040–1176. North-Holland, Amsterdam, 1998.

[HKMN08] G. Hjorth, B. Khoussainov, A. Montalbán, and A. Nies. From automatic structures to borel structures. In *LICS'08*, pages 431–441. IEEE Computer Society Press, 2008.

[Jec02] Th. Jech. *Set Theory.* Springer Monographs in Mathematics. Springer, 3rd edition, 2002.

[Joc72] C.G. Jockusch. Ramsey's theorem and recursion theory. *Journal of Symbolic Logic*, 37:268–280, 1972.

[KL08] D. Kuske and M. Lohrey. First-order and counting theories of ω-automatic structures. *Journal of Symbolic Logic*, 73:129–150, 2008.

[KN95] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *Logic and Computational Complexity*, Lecture Notes in Comp. Science vol. 960, pages 367–392. Springer, 1995.

[Ram30] F.P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc.*, 30:264–286, 1930.

[Rub08] S. Rubin. Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic*, 14:169–209, 2008.

[Sie33] W. Sierpiński. Sur un problème de la thèorie des relations. *Ann. Scuola Norm. Sup. Pisa*, 2(2):285–287, 1933.

[Sta97] L. Staiger. ω-languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages Vol. 3*, pages 339–387. Springer, 1997.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 549-560
www.stacs-conf.org

AN EFFICIENT QUANTUM ALGORITHM FOR SOME INSTANCES OF THE GROUP ISOMORPHISM PROBLEM

FRANÇOIS LE GALL

Department of Computer Science, The University of Tokyo
E-mail address: legall@is.s.u-tokyo.ac.jp

Abstract. In this paper we consider the problem of testing whether two finite groups are isomorphic. Whereas the case where both groups are abelian is well understood and can be solved efficiently, very little is known about the complexity of isomorphism testing for nonabelian groups. Le Gall has constructed an efficient classical algorithm for a class of groups corresponding to one of the most natural ways of constructing nonabelian groups from abelian groups: the groups that are extensions of an abelian group A by a cyclic group $\mathbb{Z}_m$ with the order of A coprime with m. More precisely, the running time of that algorithm is almost linear in the order of the input groups. In this paper we present a *quantum* algorithm solving the same problem in time polynomial in the *logarithm* of the order of the input groups. This algorithm works in the black-box setting and is the first quantum algorithm solving instances of the nonabelian group isomorphism problem exponentially faster than the best known classical algorithms.

1. Introduction

Testing group isomorphism (the problem asking to decide, for two given finite groups G and H, whether there exists an isomorphism between G and H) is a fundamental problem in computational group theory but little is known about its complexity. It is known that the group isomorphism problem (for groups given by their multiplication tables) reduces to the graph isomorphism problem [18], and thus the group isomorphism problem is in the complexity class $NP \cap coAM$ (since the graph isomorphism problem is in this class [2]). Miller [24] has developed a general technique to check group isomorphism in time $O(n^{\log n + O(1)})$, where n denotes the size of the input groups and Lipton, Snyder and Zalcstein [22] have given an algorithm working in $O(\log^2 n)$ space. However, no polynomial-time algorithm is known for the general case of this problem.

Another line of research is the design of algorithms solving the group isomorphism problem for particular classes of groups. For abelian groups polynomial-time algorithms follow directly from efficient algorithms for the computation of the Smith normal form of integer matrices [8, 15]. More efficient methods have been given by Vikas [28] and Kavitha [16] for

1998 ACM Subject Classification: F.2.2 Nonnumerical Algorithms and Problems.
Key words and phrases: Quantum Algorithms, Group Isomorphism Problem, Black-box Groups.
This work was done while the author was a researcher at Kyoto University, affiliated with the ERATO-SORST Quantum Computation and Information Project, Japan Science and Technology Agency.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2484

abelian groups given by their multiplication tables, and fast parallel algorithms have been constructed by McKenzie and Cook [23] for abelian permutation groups. The current fastest algorithm solving the abelian group isomorphism problem for groups given as black-boxes has been developed by Buchmann and Schmidt [5] and works in time $O(n^{1/2}(\log n)^{O(1)})$. However, as far as nonabelian groups are concerned, very little is known. For solvable groups Arvind and Torán [1] have shown that the group isomorphism problem is in $NP \cap coNP$ under certain complexity assumptions but, until recently, the only polynomial-time algorithms testing isomorphism of nontrivial classes of nonabelian groups were a result by Garzon and Zalcstein [12], which holds for a very restricted class, and a body of works initiated by Cooperman et al. [9] on simple groups.

Very recently, Le Gall [19] proposed an efficient classical algorithm solving the group isomorphism problem over another class of nonabelian groups. Since for abelian groups the group isomorphism problem can be solved efficiently, that work focused on one of the most natural next targets: cyclic extensions of abelian groups. Loosely speaking such extensions are constructed by taking an abelian group A and adding one element y that, in general, does not commute with the elements in A. More formally the class of groups considered in [19], denoted by $\mathscr{S}$, was the following.

Definition 1.1. Let G be a finite group. The group G is said to be in the class $\mathscr{S}$ if there exist a normal abelian subgroup A in G and an element $y \in G$ of order coprime with $|A|$ such that $G = \langle A, y \rangle$.

In technical words G is an extension of an abelian group A by a cyclic group $\mathbb{Z}_m$ with $gcd(|A|, m) = 1$. This class of groups includes all the abelian groups and many non-abelian groups too, as discussed in details in [19]. For example, for $A = \mathbb{Z}_3^4$ and $m = 4$, there are exactly 9 isomorphism classes in $\mathscr{S}$ (1 class of abelian groups and 8 classes of nonabelian groups). Moreover, the class $\mathscr{S}$ includes several groups that have been the target of quantum algorithms, as discussed later. The main result in [19] was the following theorem.

Theorem 1.2 ([19])**.** *There exists a deterministic algorithm checking whether two groups G and H in the class $\mathscr{S}$ (given as black-box groups) are isomorphic and, if this is the case, computing an isomorphism from G to H. Its running time has for upper bound $n^{1+o(1)}$, where $n = min(|G|, |H|)$.*

In the present paper, we focus on *quantum algorithms* solving the group isomorphism problem in the black-box setting. Cheung and Mosca [7] have shown how to compute the decomposition of an abelian group into a direct product of cyclic subgroups in time polynomial in the logarithm of its order on a quantum computer, and thus how to solve the abelian group isomorphism problem in time polynomial in $\log n$ in the black-box model. This then gives an exponential speed-up with respect to the best known classical algorithms for the same task. One can naturally ask whether a similar speed-up can be obtained for classes of nonabelian groups. In this paper, we prove that this is the case. Our main result is the following theorem.

Theorem 1.3. *There exists a quantum algorithm checking with high probability whether two groups G and H in the class $\mathscr{S}$ given as black-box groups are isomorphic and, if this is the case, computing an isomorphism from G to H. Its running time is polynomial in* $\log n$*, where $n = min(|G|, |H|)$.*

To our knowledge, this is the first quantum algorithm solving nonabelian instances of the group isomorphism problem exponentially faster than the best known classical algorithms.

Our algorithm relies on several new quantum reductions to instances of the so-called abelian Hidden Subgroup Problem, a problem that can be solved efficiently on a quantum computer. Our result can then be seen as an extension of the polynomial-time library of computational tasks which can be accomplished using Shor's factoring and discrete logarithm algorithms [27], and further quantum algorithms for abelian groups. We also mention that groups in the class $\mathscr{S}$ appear at several occasions in the quantum computation literature, mostly connected to the Hidden Subgroup Problem over semidirect product groups [4, 10, 13, 25]. Our techniques may have applications in the design of further quantum algorithms for this problem, or for other similar group-theoretic tasks.

Our quantum algorithm follows the same line as the classical algorithm in [19], but the two main technical parts are both significantly improved and modified.

Since a group G in the class $\mathscr{S}$ may in general be written as the extension of an abelian group A_1 by a cyclic group $\mathbb{Z}_{m_1}$ and as the extension of an abelian group A_2 by a cyclic group $\mathbb{Z}_{m_2}$ with $A_1 \ncong A_2$ and $m_1 \neq m_2$, we use, as in [19], the concept of a standard decomposition of G, which is an invariant for the groups in the class $\mathscr{S}$ in the sense that two isomorphic groups have similar standard decompositions (but the converse is false). A method for computing efficiently standard decompositions in the black-box model was one of the main contributions of [19], where the time complexity of this step was $O(n^{1+o(1)})$ due to the fact that the procedure proposed had to try, in the worst case, for each generator g of G, all the divisors of $|g|$. Instead, in the present work we propose a different procedure for this task (Section 3), which can be implemented in time polynomial in $\log n$ on a quantum computer, based on careful reductions to group-theoretic problems for which known efficient quantum algorithms are known: order finding, decomposing abelian groups and constructive membership in abelian groups.

Knowing standard decompositions of G and H allows us to consider only the case where H and G are two extensions of the same abelian group A by the same cyclic group $\mathbb{Z}_m$ (Proposition 6.1). Two matrices M_1 and M_2 in the group $GL(r, \mathbb{F})$ of invertible matrices of size $r \times r$ over some well-chosen finite field $\mathbb{F}$ can then be associated to the action of $\mathbb{Z}_m$ on A in the groups G and H respectively. The second main technical contribution of [19] showed that, loosely speaking, testing isomorphism of G and H then reduces (when the order of A is coprime with m) to checking whether there exists an integer $k \in \{1, \ldots, m\}$ such that M_1 and M_2^k are conjugate in $GL(r, \mathbb{F})$. The strategy adopted in [19] to solve this problem had time complexity close to n in the worst case (basically, all the integers k in $\{1, \ldots, m\}$ were checked). In the present paper, we give a poly$(\log n)$ time quantum algorithm for this problem. More generally, we show in Section 5 that the problem of testing, for any two matrices M_1 and M_2 in $GL(r, \mathbb{F})$ where r is any positive integer and $\mathbb{F}$ is any finite field, whether there exists a positive integer k such that M_1 and M_2^k are conjugate in the group $GL(r, \mathbb{F})$ reduces to solving an instance of a problem we call Set Discrete Logarithm. This quantum reduction is efficient in that it can be implemented in time polynomial in both r and $\log |\mathbb{F}|$, and works by considering field extensions of $\mathbb{F}$ and matrix invariants of M_1 and M_2.

Loosely speaking, the problem Set Discrete Logarithm asks, given two sets $\{x_1, \ldots, x_v\}$ and $\{y_1, \ldots, y_v\}$ of elements in $\mathbb{F}$, to compute an integer k such that $\{y_1^k, \ldots, y_v^k\} = \{x_1, \ldots, x_v\}$, if such an integer exists. This computational problem is a generalization of the standard discrete logarithm problem (which is basically the case $v = 1$) but appears to be much more challenging. The quantum algorithm we propose (in Section 4) works in time polynomial in v and $\log |\mathbb{F}|$, and relies on a reduction to several instances of the abelian

Hidden Subgroup Problem. Our solution to the problem Set Discrete Logarithm is then an extension of the computational tasks which can be solved efficiently using known quantum algorithms for abelian groups.

2. Preliminaries

2.1. Group theory and standard decompositions

We assume that the reader is familiar with the basic notions of group theory and state without proofs definitions and properties of groups we will use in this paper.

For any positive integer m, we denote by $\mathbb{Z}_m$ the additive cyclic group of integers $\{0,\ldots,m-1\}$, and by $\mathbb{Z}_m^*$ the multiplicative group of integers in $\{1,\ldots,m-1\}$ coprime with m.

Let G be a finite group. For any subgroup H and any normal subgroup K of G we denote by HK the subgroup $\{hk \mid h\in H, k\in K\} = \{kh \mid h \in H, k\in K\}$. Given a set S of elements of G, the subgroup generated by the elements of S is written $\langle S\rangle$. We say that two elements g_1 and g_2 of G are conjugate in G if there exists an element $y\in G$ such that $g_2 = yg_1y^{-1}$. For any two elements $g,h\in G$ we denote by $[g,h]$ the commutator of g and h, i.e., $[g,h] = ghg^{-1}h^{-1}$. More generally, given two subsets S_1 and S_2 of G, we define $[S_1,S_2] = \langle [s_1,s_2] \mid s_1\in S_1, s_2\in S_2\rangle$. The commutator subgroup of G is defined as $G' = [G,G]$. The derived series of G is defined recursively as $G^{(0)} = G$ and $G^{(i+1)} = (G^{(i)})'$. The group G is said to be solvable if there exists some integer k such that $G^{(k)} = \{e\}$. Given two groups G_1 and G_2, a map $\phi : G_1 \to G_2$ is a homomorphism from G_1 to G_2 if, for any two elements g and g' in G_1, the relation $\phi(gg') = \phi(g)\phi(g')$ holds. We say that G_1 and G_2 are isomorphic if there exists a one-one homomorphism from G_1 to G_2, and we write $G_1 \cong G_2$.

Given any finite group G, we denote by $|G|$ its order and, given any element g in G, we denote by $|g|$ the order of g in G. For any prime p, we say that a group is a p-group if its order is a power of p. If $|G| = p_1^{e_i}\ldots p_r^{e_r}$ for distinct prime numbers p_i, then for each $i\in\{1,\ldots,r\}$ the group G has a subgroup of order $p_i^{e_i}$. Such a subgroup is called a Sylow p_i-subgroup of G. Moreover, if G is additionally abelian, then each Sylow p_i-group is unique and G is the direct product of its Sylow subgroups. Abelian p-groups have remarkably simple structures: any abelian p-group is isomorphic to a direct product of cyclic p-groups $\mathbb{Z}_{p^{f_1}}\times\cdots\times\mathbb{Z}_{p^{f_s}}$ for some positive integer s and positive integers $f_1 \leq \ldots \leq f_s$, and this decomposition is unique. We say that a set $\{g_1,\ldots,g_t\}$ of elements of an abelian group G is a basis of G if $G = \langle g_1\rangle\times\cdots\times\langle g_t\rangle$ and the order of each g_i is a prime power.

For a given group G in the class $\mathscr{S}$ in general many different decompositions as an extension of an abelian group by a cyclic group exist. For example, the abelian group $\mathbb{Z}_6 = \langle x_1,x_2 \mid x_1^2 = x_2^3 = [x_1,x_2] = e\rangle$ can be written as $\langle x_1\rangle\times\langle x_2\rangle$, $\langle x_2\rangle\times\langle x_1\rangle$ or $\langle x_1,x_2\rangle\times\{e\}$. That is why we introduce the notion of a standard decomposition, as it was done in [19].

Definition 2.1. Let G be a finite group in the class $\mathscr{S}$. For any positive integer m denote by $\mathscr{D}_G^m$ the set (possibly empty) of pairs (A,B) such that the following three conditions hold: (i) A is a normal abelian subgroup of G of order coprime with m; and (ii) B is a cyclic subgroup of G of order m; and (iii) $G = AB$. Let $\gamma(G)$ be the smallest positive integer such that $\mathscr{D}_G^{\gamma(G)}\neq\varnothing$. A standard decomposition of G is an element of $\mathscr{D}_G^{\gamma(G)}$.

2.2. Black-box groups

In this paper we work in the black-box model. A black-box group is a representation of a group G where elements are represented by strings, and an oracle is available to perform group operations. To be able to take advantage of the power of quantum computation when dealing with black-box groups, the oracles performing group operations have to be able to deal with quantum superpositions. These quantum black-box groups have been first studied by Ivanyos et al. [14] and Watrous [29, 30], and have become the standard model for studying group-theoretic problems in the quantum setting.

More precisely, a quantum black-box group is a representation of a group where elements are represented by strings (of the same length, supposed to be logarithmic in the order of the group). We assume the usual unique encoding hypothesis, i.e., each element of the group is encoded by a unique string, which is crucial for technical reasons (without it, most quantum algorithms do not work). A quantum oracle V_G is available, such that $V_G(|g\rangle|h\rangle) = |g\rangle|gh\rangle$ for any g and h in G (using strings to represent the group elements), and behaving in an arbitrary way on other inputs. We say that a group G is input as a black-box if a set of strings representing generators $\{g_1, \ldots, g_s\}$ of G with $s = O(\log|G|)$ is given as input, and queries to the oracle can be done at cost 1. The hypothesis on s is natural since every group G has a generating set of size $O(\log|G|)$, and enables us to make the exposition of our results easier. Also notice that a set of generators of any size can be converted efficiently into a set of generators of size $O(\log|G|)$ if randomization is allowed.

Any efficient quantum black-box algorithm gives rise to an efficient concrete quantum algorithm whenever the oracle operations can be replaced by efficient procedures. Especially, when a mathematical expression of the generators input to the algorithm is known, performing group operations can be done directly on the elements in polynomial time (in $\log|G|$) for many natural groups, including permutation groups and matrix groups.

Quantum algorithms are very efficient for solving computational problems over abelian groups. In the following theorem, we describe the main results we will need in this paper.

Theorem 2.2 ([7, 14, 27])**.** *There exists quantum algorithms solving, in time polynomial in* $\log|G|$*, the following computational tasks with probability at least* $1 - 1/poly(|G|)$*:*

(i) *Given a group G given as a black-box (with unique encoding) and any element $g \in G$, compute the order of g in G.*

(ii) *Given an abelian group G given as a black-box (with unique encoding), compute a basis $(g_1, \ldots, g_s)$ of G.*

(iii) *Given an abelian group G given as a black-box (with unique encoding), a basis $(g_1, \ldots, g_s)$ of G, and any $g \in G$, compute a decomposition of g over $(g_1, \ldots, g_s)$, i.e., integers $u_1, \ldots, u_s$ such that $g = g_1^{u_1} \cdots g_s^{u_s}$.*

Actually, all the tasks in Theorem 2.2 can be seen as black-boxes versions of instances of the so-called Hidden Subgroup Problem (HSP) over abelian groups. It is known that the abelian HSP can be solved in time polynomial in $\log|G|$ [17], even if G is given as a black-box group with unique encoding [14, 26].

3. Computing a Standard Decomposition

In this section we present a quantum algorithm computing a standard decomposition of any group in the class $\mathscr{S}$ in time polynomial in the logarithm of the order of the group.

The precise description of the algorithm, which we denote Procedure Decompose, is given in metacode in Figure 1. Further descriptions on how each step is implemented follow.

```
Procedure Decompose
    INPUT: a set of generators $\{g_1, \ldots, g_s\}$ of a group $G$ in $\mathscr{S}$ with $s = O(\log|G|)$.
    OUTPUT: a pair $(U, v)$ where $U$ is a subset of $G$ and $v \in G$.
 1  compute generators $\{g'_1, \ldots, g'_t\}$ of the derived subgroup $G'$ with $t = O(\log|G|)$;
 2  compute $\kappa = lcm(|g_1|, \ldots, |g_s|)$;
 3  factorize $\kappa$ and write $\kappa = p_1^{e_1} \cdots p_r^{e_r}$ where the prime numbers $p_i$ are distinct;
 4  $U \leftarrow \{g'_1, \ldots, g'_t\}$; $V \leftarrow \varnothing$; $\Sigma \leftarrow \varnothing$;
 5  for $i = 1$ to $r$
 6      do
 7          $\Gamma_i \leftarrow \varnothing$;
 8          for $j = 1$ to $s$ do $\Gamma_i \leftarrow \Gamma_i \cup \{g_j^{\kappa/p_i^{e_i}}\}$;
 9          if $[\Gamma_i, G'] = e$ and $gcd(p_i, |G'|) \neq 1$ then $U \leftarrow U \cup \Gamma_i$;
10          if $[\Gamma_i, G'] = e$ and $gcd(p_i, |G'|) = 1$
11                      then
12                          search for an element $\gamma_i \in \Gamma_i$ such that $\langle\Gamma_i\rangle G' = \langle\gamma_i, G'\rangle$;
13                          if no such element exists
14                              then $U \leftarrow U \cup \Gamma_i$
15                              else $\Sigma \leftarrow \Sigma \cup \{\gamma_i\}$;
16                      endthen
17          if $[\Gamma_i, G'] \neq e$ then { take an element $\gamma_i \in \Gamma_i$ such that $|\gamma_i| = \max_{\gamma\in\Gamma_i}|\gamma|$;
18                                  $V \leftarrow V \cup \{\gamma_i\}$; }
19      enddo
20  for all $w$ in $\Sigma$
21      do
22        if there exists an element $z$ in $\Sigma$ such that $[w, z] \neq e$
23            then { if $zwz^{-1} \in \langle w\rangle$ then $U \leftarrow U \cup \{w\}$ else $V \leftarrow V \cup \{w\}$; }
24      enddo
25  for all $w \in \Sigma\backslash(U \cup V)$
26      do
27        if $[w, u] = \{e\}$ for all $u \in U$ then $U \leftarrow U \cup \{w\}$ else $V \leftarrow V \cup \{w\}$;
28      enddo
29  $b \leftarrow \Pi_{g\in V}|g|$; $z \leftarrow \Pi_{g\in V}g$; $v \leftarrow z^{|z|/b}$;
30  output $(U, v)$;
```

Figure 1: Procedure Decompose.

- At Step 1 a set of generators $\{g'_1, \ldots, g'_t\}$ of the derived subgroup G' with $t = O(\log|G|)$ is computed in time polynomial in $\log|G|$ with success probability $1 - 1/\text{poly}(|G|)$ using the classical algorithm by Babai et al. [3].
- The order of G' at Steps 9 and 10, and the orders of elements at Steps 2, 17 and 29 are computed using the quantum algorithms for Tasks (i) and (ii) in Theorem 2.2.
- The least common multiple at Step 2 is computed using standard algorithms, and is factorized at Step 3 using Shor's factoring algorithm [27].

- At Step 12, notice that $[\Gamma_i, G'] = e$ implies that $\langle \Gamma_i \rangle G'$ is an abelian group. For each element γ_i in Γ_i (there are $O((\log |G|)^2)$ such elements), the quantum algorithms for Tasks (i) and (ii) in Theorem 2.2 are used to check whether $|\langle \Gamma_i \rangle G'| = |\langle \gamma_i, G' \rangle|$. Since necessarily $\langle \gamma_i, G' \rangle \leq \langle \Gamma_i \rangle G'$, this test is sufficient to check whether $\langle \Gamma_i \rangle G' = \langle \gamma_i, G' \rangle$.
- The tests at Steps 9, 10 to 17 are done by noticing that $[\Gamma_i, G'] = \{e\}$ if and only if $[\gamma, g'_j] = e$ for each $\gamma \in \Gamma_i$ and each $j \in \{1, \ldots, t\}$.
- Testing whether zwz^{-1} is in $\langle w \rangle$ at Step 23 is done by trying to decompose zwz^{-1} over $\langle w \rangle$ using the quantum algorithm for Task (iii) in Theorem 2.2, and then checking if the decomposition indeed represents zwz^{-1} (since, a priori, this algorithm can have an arbitrary behavior when $zwz^{-1} \notin \langle w \rangle$).

This description, along with Theorem 2.2 and with the observation that the sets U, V and Σ have size $O((\log |G|)^2)$, show that all the steps of Procedure DECOMPOSE can be implemented in time polynomial in $\log |G|$. The following theorem states its correctness.

Theorem 3.1. *Let G be a group in the class $\mathscr{S}$, given as a black-box group (with unique encoding). The procedure* DECOMPOSE *on input G outputs, with high probability, a pair (U, v) such that $(\langle U \rangle, \langle v \rangle)$ is a standard decomposition of G. It can be implemented in time polynomial in* $\log |G|$ *on a quantum computer.*

A complete proof of Theorem 3.1 can be found in the full version of this paper [20].

4. Set Discrete Logarithm

We first introduce the following useful notation. Let $\mathbb{F}$ be a finite field, and $\Sigma = \{x_1, \ldots, x_t\}$ be any subset of $\mathbb{F}$ with possible repetitions, i.e., all the x_i's are elements of $\mathbb{F}$, but may not be distinct. For any integer k, we denote by Σ^k the subset of $\mathbb{F}$ with possible repetitions $\{x_1^k, \ldots, x_t^k\}$.

In this section we consider the following problem. Here u is a positive integer which is a parameter of the problem (taking $u \geq 2$ does not make the problem significantly harder, but this enables us to give a more convenient presentation of our results).

SET DISCRETE LOGARITHM
INPUT: two lists $(S_1, \ldots, S_u)$ and $(T_1, \ldots, T_u)$ where, for each integer $h \in \{1, \ldots, u\}$,
S_h and T_h are subsets with possible repetitions of some finite field $\mathbb{F}_h$.
OUTPUT: a positive integer k such that $T_h^k = S_h$ for all $h \in \{1, \ldots, u\}$, if such k exists.

Notice that the case $u = 1$ with $|S_1| = |T_1| = 1$ is the usual discrete logarithm problem over the multiplicative group of the field $\mathbb{F}_1$. Actually, our algorithm solving the problem SET DISCRETE LOGARITHM will only need the multiplicative structure of the fields, and then also works if we replace in the definition each field $\mathbb{F}_h$ by any multiplicative finite group G_h. However, since the main applications of our algorithm deal with field structures, we describe our results in the present slightly less general form.

Given an instance of SET DISCRETE LOGARITHM, let m_S denote the smallest positive integer such that $x^{m_S} = 1$ for all $x \in S_1 \cup \cdots \cup S_u$, and let m_T denote the smallest positive integer such that $y^{m_T} = 1$ for all $y \in T_1 \cup \cdots \cup T_u$. The main result of this section is the following theorem.

Theorem 4.1. *There exists a quantum algorithm that solves with high probability the problem* Set Discrete Logarithm*, and runs in time polynomial in u, $\log(m_S + m_T)$, and* $\max_{1\leq h\leq u}(|S_h| + |T_h| + \log|\mathbb{F}_h|)$.

Proof. For the sake of brevity, let us denote $\Sigma = S_1 \cup \cdots \cup S_u \cup T_1 \cup \cdots \cup T_u$. We first compute the orders of all the elements in Σ using Shor's algorithm [27]. The value m_S is the least common multiple of the orders of all the elements in $S_1 \cup \cdots \cup S_u$, and the value m_T is the least common multiple of the orders of all the elements in $T_1 \cup \cdots \cup T_u$. The values m_S and m_T can then be computed in time polynomial in $\log(m_S + m_T)$, $|\Sigma|$, and $\max_{1\leq h\leq u}\log|\mathbb{F}_h|$. Notice that, for any positive integer k, the least common multiple of the orders of all the elements in $T_1^k \cup \cdots \cup T_u^k$ is $m_T/gcd(k, m_T)$. Then, if m_S does not divide m_T, there is no solution to the problem Set Discrete Logarithm. If m_S divides m_T but $m_S \neq m_T$, then a solution (if it exists) can be found by replacing the list $(T_1, \ldots, T_u)$ by the list $(T_1^{m_T/m_S}, \ldots, T_u^{m_T/m_S})$. Thus, without loss of generality, we suppose hereafter that $m_S = m_T$ and denote by m this value. Then a solution k can be searched for in the set $\mathbb{Z}_m^*$.

Let $\{m_1, \ldots, m_\ell\} = \cup_{z\in\Sigma}\{|z|\}$ denote the set of orders of the elements in Σ. For each $h \in \{1, \ldots, u\}$ and each $i \in \{1, \ldots, \ell\}$, we define the subsets

$$S_{h,i} = \{x \in S_h \mid |x| = m_i\} \text{ and } T_{h,i} = \{y \in T_h \mid |y| = m_i\}.$$

Let us also define the sets

$$K_{h,i} = \{k \in \mathbb{Z}_m^* \mid T_{h,i}^k = S_{h,i}\} \text{ and } \overline{K}_{h,i} = \{k \in \mathbb{Z}_m^* \mid T_{h,i}^k = T_{h,i}\}.$$

It is straightforward to check that the set $\overline{K}_{h,i}$ is a subgroup of $\mathbb{Z}_m^*$, and that the set $K_{h,i}$ is either empty, or is a coset of $\overline{K}_{h,i}$ in $\mathbb{Z}_m^*$.

Let $K \subseteq \mathbb{Z}_m^*$ denote the set of solutions of the instance of Set Discrete Logarithm we are considering. Then

$$K = \bigcap_{1\leq h\leq u}\Big(\bigcap_{1\leq i\leq \ell} K_{h,i}\Big).$$

The set K can be computed efficiently using a quantum computer if, for each $h \in \{1, \ldots, u\}$ and each $i \in \{1, \ldots, \ell\}$, the set $K_{h,i}$ is known. More precisely, this is done by using a quantum algorithm for computing the intersections of two cosets of an abelian group — more details can be found in the full version of this paper [20].

The final part of the proof shows how to compute these sets $K_{h,i}$. Let us fix an integer $h \in \{1, \ldots, u\}$ and an integer $i \in \{1, \ldots, \ell\}$. We suppose that $S_{h,i}$ and $T_{h,i}$ have the same size (otherwise $K_{h,i} = \varnothing$ and thus $K = \varnothing$). Denote $S_{h,i} = \{x_1, \ldots, x_v\}$ and $T_{h,i} = \{y_1, \ldots, y_v\}$, where $v = |S_{h,i}|$ depends on h and i. We present a quantum procedure computing a set of generators of $\overline{K}_{h,i}$, and an element $k_{h,i}$ in $K_{h,i}$ when this set is not empty, in time polynomial in v, $\log m$, and $\log|\mathbb{F}_h|$.

We first show how to compute the subgroup $\overline{K}_{h,i}$. Let $\prec$ be an arbitrary strict total ordering of the elements of $\mathbb{F}_h$. Without loss of generality we can suppose that $x_1 \preceq x_2 \preceq \cdots \preceq x_v$. Let μ be the function from $\mathbb{Z}_m^* \times \{1, \ldots, v\}$ to $\mathbb{F}_h$ defined as follows: for any $k \in \mathbb{Z}_m^*$ and any $j \in \{1, \ldots, v\}$, $\mu(k, j)$ is the j-th element (with respect to the order $\prec$) of the set $T_{h,i}^k$. Let f be the function from $\mathbb{Z}_m^*$ to $(\mathbb{F}_h)^v$ such that, for any $k \in \mathbb{Z}_m^*$:

$$f(k) = (\mu(k, 1)y_1^{-1}, \ldots, \mu(k, v)y_v^{-1}).$$

Notice that the set $\{k \in \mathbb{Z}_m^* \mid f(k) = (1,\ldots,1)\}$ is precisely the subgroup $\overline{K}_{h,i}$ of $\mathbb{Z}_m^*$. Moreover, the function f is constant on cosets of $\overline{K}_{h,i}$ in $\mathbb{Z}_m^*$, with distinct values on distinct cosets (since $f(k_1) = f(k_2)$ implies that $T_{h,i}^{k_1} = T_{h,i}^{k_2}$ and thus $k_1 \in k_2\overline{K}_{h,i}$). This is thus an instance of the abelian HSP, and a set of generators of $\overline{K}_{h,i}$ can be found in time polynomial in v, $\log m$ and $\log|\mathbb{F}_h|$ using the algorithm described in Subsection 2.2 (notice that the underlying group is $\mathbb{Z}_m^*$, and that the value of the function f can be computed in time v, $\log m$ and $\log|\mathbb{F}_h|$).

We now show how to compute an element $k_{h,i}$ in $K_{h,i}$ if this set is not empty. We first try to find an element $\alpha \in \mathbb{Z}_{m_i}^*$ such that $T_{h,i}^{\alpha} = S_{h,i}$. This is done by, for each $j \in \{1,\ldots,v\}$, trying to find an integer $\alpha_j \in \mathbb{Z}_{m_i}^*$ such that $x_1^{\alpha_j} = y_j$, if such an integer exists (notice that, for each j, there is at most one element α_j in $\mathbb{Z}_{m_i}^*$ satisfying this condition, which can be computed in time polynomial in $\log m_i$ and $\log|\mathbb{F}_h|$ using the quantum algorithm for the standard discrete logarithm problem [27]) and checking whether $T_{h,i}^{\alpha_j} = S_{h,i}$. If no such value α can be found, we conclude that $K_{h,i}$ is empty. Otherwise we take any such value α and compute $k_{h,i}$ as follows. Let us write the prime power decomposition of m as $m = p_1^{\epsilon_1}\cdots p_r^{\epsilon_r}p'^{\eta_1}_1\cdots p'^{\eta_s}_s q_1^{\delta_1}\cdots q_t^{\delta_t}$, where each prime p_l divides m_i for $l \in \{1,\ldots,r\}$, each prime p'_l divides α but not m_i for $l \in \{1,\ldots,s\}$, and each prime q_l divides neither m_i nor α for $l \in \{1,\ldots,t\}$. Then the integer

$$k_{h,i} = \alpha + m_i q_1^{\delta_1}\cdots q_t^{\delta_t} \bmod m$$

is coprime with m (since α is coprime with m_i and then each prime p_l, p'_l or q_l does not divide $k_{h,i}$), and hence is in $\mathbb{Z}_m^*$. From the choice of α and since any element in $T_{h,i}$ has order m_i, we conclude that $k_{h,i}$ is in the set $K_{h,i}$. ■

5. Discrete Logarithm up to Conjugacy

Given a positive integer r and a finite field $\mathbb{F}$, remember that $GL(r,\mathbb{F})$ denotes the multiplicative group of invertible matrices of size $r \times r$ with entries in $\mathbb{F}$. In this section we consider the following problem. Here u is again a positive integer which is a parameter of the problem.

Discrete Log up to Conjugacy

input: two lists of matrices $(M_1^{(1)},\ldots,M_1^{(u)})$ and $(M_2^{(1)},\ldots,M_2^{(u)})$ where, for each integer $h \in \{1,\ldots,u\}$, $M_1^{(h)}$ and $M_2^{(h)}$ are in $GL(r_h,\mathbb{F}_h)$ for some positive integer r_h and some finite field $\mathbb{F}_h$.

output: a positive integer k and u matrices $M^{(h)} \in GL(r_h,\mathbb{F}_h)$ such that $M^{(h)} \cdot M_1^{(h)} = [M_2^{(h)}]^k \cdot M^{(h)}$ for each $h \in \{1,\ldots,u\}$, if such elements exist.

In the statement of the above problem, the notation $[M_2^{(h)}]^k$ simply means $M_2^{(h)}$ raised to the k-th power. Notice that the case $u = 1$ and $r_1 = 1$ is basically the usual discrete logarithm problem over the multiplicative group of the finite field $\mathbb{F}_1$.

Let m_1 and m_2 denote the smallest positive integers such that $[M_1^{(h)}]^{m_1} = I$ and $[M_2^{(h)}]^{m_2} = I$ for all $h \in \{1,\ldots,u\}$. The main result of this section is the following theorem.

Theorem 5.1. *There exists a quantum algorithm that solves with high probability the problem* Discrete Log up to Conjugacy, *and runs in time polynomial in u,* $\log(m_1+m_2)$, *and* $\max_{1\leq h\leq u}(r_h+\log|\mathbb{F}_h|)$

The quantum algorithm solving the problem Discrete Log up to Conjugacy follows from an efficient reduction to the problem Set Discrete Logarithm, using the concepts of elementary divisors, Jordan normal forms and similarity of matrices. A complete proof of Theorem 5.1 is given in the full version of this paper [20].

6. Proof of Theorem 1.3

We will need the following result from [19] that shows necessary and sufficient conditions for the isomorphism of two groups in the class $\mathscr{S}$.

Proposition 6.1 (Proposition 5.1 in [19])**.** *Let G and H be two groups in $\mathscr{S}$. Let $(A_1,\langle y_1\rangle)$ and $(A_2,\langle y_2\rangle)$ be standard decompositions of G and H respectively and let $\varphi_1\in Aut(A_1)$ (resp. $\varphi_2\in Aut(A_2)$) be the action by conjugation of y_1 on A_1 (resp. of y_2 on A_2). The groups G and H are isomorphic if and only if the following three conditions hold: (i) $A_1\cong A_2$; and (ii) $|y_1|=|y_2|$; and (iii) there exist a positive integer k and an isomorphism $\chi\colon A_1\to A_2$ such that $\varphi_1=\chi^{-1}\varphi_2^k\chi$, where φ_2^k means φ_2 composed by itself k times.*

We now present our proof of Theorem 1.3.

Proof of Theorem 1.3. Suppose that G and H are two groups in the class $\mathscr{S}$. In order to test whether these two groups are isomorphic, we first run Procedure Decompose on G and H and obtain outputs (U_1,y_1) and (U_2,y_2) such that $(\langle U_1\rangle,\langle y_1\rangle)$ and $(\langle U_2\rangle,\langle y_2\rangle)$ are standard decompositions of G and H respectively with high probability (from Theorem 3.1). The running time of this step is polynomial in the logarithms of $|G|$ and $|H|$, from Theorem 3.1. Denote $A_1=\langle U_1\rangle$ and $A_2=\langle U_2\rangle$. The orders of A_1,A_2, y_1 and y_2 are then computed using the quantum algorithms for Tasks (i) and (ii) in Theorem 2.2. Notice that $|G|=|A_1|\cdot|y_1|$ and $|H|=|A_2|\cdot|y_2|$. If $|G|\neq|H|$, we conclude that G and H are not isomorphic. In the following, we suppose that $|G|=|H|$ and denote by n this order.

If $|y_1|\neq|y_2|$ we conclude that G and H are not isomorphic, from Proposition 6.1. Otherwise denote $|y_1|=|y_2|=m$. Then we compute a basis $(g_1,\ldots,g_s)$ of A_1 and a basis $(h_1,\ldots,h_{s'})$ of A_2 using the quantum algorithm for Task (ii) in Theorem 2.2. Given these bases it is easy to check the isomorphism of A_1 and A_2: the groups A_1 and A_2 are isomorphic if and only if $s=s'$ and there exists a permutation σ of $\{1,\ldots,s\}$ such that $|g_i|=|h_{\sigma(i)}|$ for each $i\in\{1,\ldots,s\}$. If $A_1\ncong A_2$ we conclude that G and H are not isomorphic, from Proposition 6.1.

Now suppose that $A_1\cong A_2\cong(\mathbb{Z}_{p_1^{f_1}})^{r_1}\times\cdots\times(\mathbb{Z}_{p_t^{f_t}})^{r_t}$, where each p_i is a prime, but $p_i^{f_i}\neq p_j^{f_j}$ for $i\neq j$. We want to decide whether the action by conjugation $\varphi_1\in Aut(A_1)$ of y_1 on A_1 and the action by conjugation $\varphi_2\in Aut(A_2)$ of y_2 on A_2 satisfy Condition (iii) in Proposition 6.1. Notice that, for each $j\in\{1,\ldots,s\}$, we can compute (in time polynomial in $\log n$) integers u_{ij} and v_{ij} such that $\varphi_1(g_j)=y_1g_jy_1^{-1}=g_1^{u_{1j}}\cdots g_s^{u_{sj}}$ and $\varphi_2(h_j)=y_2h_jy_2^{-1}=h_1^{v_{1j}}\cdots h_s^{v_{sj}}$ using the quantum algorithm for Task (iii) in Theorem 2.2.

Denote $\mathsf{V}=GL(r_1,\mathbb{Z}_{p_1})\times\cdots\times GL(r_t,\mathbb{Z}_{p_t})$. The theory developed in [19] shows that we can compute efficiently two elements M_1 and M_2 in V satisfying the following two conditions:

(a) $M_1^m=M_2^m=I$; and

(b) for each integer k, M_1 and M_2^k are conjugate in the group V if and only if there exists an isomorphism $\chi\colon A_1 \to A_2$ such that $\varphi_1 = \chi^{-1}\varphi_2^k\chi$.

If we denote $M_1 = (M_1^{(1)}, \ldots, M_1^{(t)})$ and $M_2 = (M_2^{(1)}, \ldots, M_2^{(t)})$, where each $M_1^{(\ell)}$ and each $M_2^{(\ell)}$ are matrices in $GL(r_\ell, \mathbb{Z}_{p_\ell})$, then checking if the later condition holds becomes an instance of the problem Discrete Log up to Conjugacy, and can be solved using the algorithm of Theorem 5.1 in time polynomial in t, $\log m$, and $\max_{1\leq\ell\leq t}(r_\ell + \log p_\ell)$, i.e., in time polynomial in $\log n$.

If the above instance of Discrete Log up to Conjugacy has no solution, we conclude that G and H are not isomorphic. Otherwise we take one value k such that M_1 and M_2^k are conjugate, along with an element $X \in \mathsf{V}$ such that $XM_1 = M_2^kX$ (such an element is obtained from the output of the algorithm of Theorem 5.1), and compute an isomorphism χ from A_1 to A_2 such that $\varphi_1 = \chi^{-1}\varphi_2^k\chi$. The map $\mu : G \to H$ defined as $\mu(xy_1^j) = \chi(x)y_2^{kj}$ for any $x \in A_1$ and any $j \in \{0, \ldots, m-1\}$ is then an isomorphism from G to H — more details on this construction can be found in the full version of this paper [20]. ■

Acknowledgments

The author is indebted to Yoshifumi Inui for many discussions on similar topics. He also thanks Erich Kaltofen, Igor Shparlinski and Yuichi Yoshida for helpful comments.

References

[1] Arvind, V., and Torán, J. Solvable group isomorphism. In *Proceedings of the 19th IEEE Conference on Computational Complexity* (2004), pp. 91–103.

[2] Babai, L. Trading group theory for randomness. In *Proceedings of the 17th annual ACM Symposium on Theory of Computing* (1985), pp. 421–429.

[3] Babai, L., Cooperman, G., Finkelstein, L., Luks, E. M., and Seress, Á. Fast Monte Carlo algorithms for permutation groups. *Journal of Computer and System Sciences 50*, 2 (1995), 296–308.

[4] Bacon, D., Childs, A. M., and van Dam, W. From optimal measurement to efficient quantum algorithms for the hidden subgroup problem over semidirect product groups. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science* (2005), pp. 469–478.

[5] Buchmann, J., and Schmidt, A. Computing the structure of a finite abelian group. *Mathematics of Computation 74*, 252 (2005), 2017–2026.

[6] Cantor, D., and Zassenhaus, H. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation 36* (1981), 587–592.

[7] Cheung, K., and Mosca, M. Decomposing finite abelian groups. *Quantum Information and Computation 1*, 3 (2001), 26–32.

[8] Chou, T.-W. J., and Collins, G. E. Algorithms for the solution of systems of linear diophantine equations. *SIAM Journal on Computing 11*, 4 (1982), 687–708.

[9] Cooperman, G., Finkelstein, L., and Linton, S. Recognizing $GL_n(2)$ in non-standard representation. In *Groups and Computation II, Proceedings of a SIMACS Workshop* (1997), pp. 85–100.

[10] Ettinger, M., and Høyer, P. On quantum algorithms for noncommutative hidden subgroups. *Advances in Applied Mathematics 25*, 3 (2000), 239–251.

[11] Friedl, K., Ivanyos, G., Magniez, F., Santha, M., and Sen, P. Hidden translation and orbit coset in quantum computing. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing* (2003), pp. 1–9.

[12] Garzon, M. H., and Zalcstein, Y. On isomorphism testing of a class of 2-nilpotent groups. *Journal of Computer and System Sciences 42*, 2 (1991), 237–248.

[13] Inui, Y., and Le Gall, F. Efficient quantum algorithms for the hidden subgroup problem over a class of semi-direct product groups. *Quantum Information and Computation 7*, 5&6 (2007), 559–570.

[14] Ivanyos, G., Magniez, F., and Santha, M. Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem. *International Journal of Foundations of Computer Science 14*, 5 (2003), 723–740.

[15] Kannan, R., and Bachem, A. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM Journal on Computing 8*, 4 (1979), 499–507.

[16] Kavitha, T. Linear time algorithms for abelian group isomorphism and related problems. *Journal of Computer and System Sciences 73*, 6 (2007), 986–996.

[17] Kitaev, A. Y. Quantum measurements and the abelian stabilizer problem. arXiv.org e-Print archive, arXiv:quant-ph/9511026, 1995.

[18] Köbler, J., Torán, J., and Schöning, U. *The graph isomorphism problem: its structural complexity.* Birkhäuser, 1993.

[19] Le Gall, F. Efficient isomorphism testing for a class of group extensions. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science* (2009), pp. 625–636. Full version available at http://arxiv.org/abs/0812.2298.

[20] Le Gall, F. An efficient quantum algorithm for some instances of the group isomorphism problem. Full version of the present paper. Available at http://arxiv.org/abs/1001.0608.

[21] Lidl, R., and Niederreiter, H. *Finite fields*. Cambridge University Press, 2008.

[22] Lipton, R. J., Snyder, L., and Zalcstein, Y. The complexity of word and isomorphism problems for finite groups. Tech. rep., John Hopkins, 1976.

[23] McKenzie, P., and Cook, S. A. The parallel complexity of abelian permutation group problems. *SIAM Journal on Computing 16*, 5 (1987), 880–909.

[24] Miller, G. On the $n^{\log n}$ isomorphism technique. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing* (1978), pp. 51–58.

[25] Moore, C., Rockmore, D. N., Russell, A., and Schulman, L. J. The power of basis selection in fourier sampling: hidden subgroup problems in affine groups. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2004), pp. 1113–1122.

[26] Mosca, M. *Quantum Computer Algorithms*. PhD thesis, Oxford university, 1999.

[27] Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing 26*, 5 (1997), 1484–1509.

[28] Vikas, N. An $O(n)$ algorithm for Abelian p-group isomorphism and an $O(n \log n)$ algorithm for Abelian group isomorphism. *Journal of Computer and System Sciences 53*, 1 (1996), 1–9.

[29] Watrous, J. Succinct quantum proofs for properties of finite groups. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science* (2000), pp. 537–546.

[30] Watrous, J. Quantum algorithms for solvable groups. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing* (2001), pp. 60–67.

TREEWIDTH REDUCTION FOR CONSTRAINED SEPARATION AND BIPARTIZATION PROBLEMS

DÁNIEL MARX [1] AND BARRY O'SULLIVAN [2] AND IGOR RAZGON [2]

[1] Tel Aviv University
E-mail address: dmarx@cs.bme.hu

[2] Cork Constraint Computation Centre, University College Cork
E-mail address: {b.osullivan,i.razgon}@cs.ucc.ie

ABSTRACT. We present a method for reducing the treewidth of a graph while preserving all the minimal $s - t$ separators. This technique turns out to be very useful for establishing the fixed-parameter tractability of constrained separation and bipartization problems. To demonstrate the power of this technique, we prove the fixed-parameter tractability of a number of well-known separation and bipartization problems with various additional restrictions (e.g., the vertices being removed from the graph form an independent set). These results answer a number of open questions in the area of parameterized complexity.

1. Introduction

Finding cuts and separators is a classical topic of combinatorial optimization and in recent years there has been an increase in interest in the fixed-parameter tractability of such problems [19, 11, 15, 28, 16, 13, 5, 20]. Recall that a problem is *fixed-parameter tractable* (or FPT) with respect to a parameter k if it can be solved in time $f(k) \cdot n^{O(1)}$ for some function $f(k)$ depending only on k [10, 12, 21]. In typical parameterized separation problems, the parameter k is the size of the separator we are looking for, thus fixed-parameter tractability with respect to this parameter means that the combinatorial explosion is restricted to the size of the separator, but otherwise the running time depends polynomially on the size of the graph.

The main technical contribution of the present paper is a theorem stating that given a graph G, two terminal vertices s and t, and a parameter k, we can compute in a FPT-time a graph G^* having its treewidth bounded by a function of k while (roughly speaking) preserving all the minimal $s - t$ separators of size at most k. Combining this theorem with the well-known Courcelle's Theorem, we obtain a powerful tool for proving the fixed parameter tractability of constrained separation and bipartization problems. We demonstrate the power of the methodology with the following results.

- We prove that the MINIMUM STABLE $s - t$ CUT problem (Is there an independent set S of size at most k whose removal separates s and t?) is fixed-parameter tractable. This problem

1998 ACM Subject Classification: G.2.2. Graph Theory, Subject: Graph Algorithms.
Key words and phrases: fixed-parameter algorithms, graph separation problems, treewidth.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2458

received some attention in the community. Our techniques allow us to prove various generalizations of this result very easily. First, instead of requiring that S is independent, we can require that it induces a graph that belongs to a hereditary class $\mathcal{G}$; the problem remains FPT. Second, in the MULTICUT problem a list of pairs of terminals are given (s_1, t_1), ..., (s_ℓ, t_ℓ) and the solution S has to be a set of at most k vertices that induces a graph from $\mathcal{G}$ and separates s_i from t_i for every i. We show that this problem is FPT parameterized by k and ℓ, which is a very strong generalization of previous results [19, 28]. Third, the results generalize to the MULTICUT-UNCUT problem, where two sets T_1, T_2 of pairs of terminals are given, and S has to separate every pair of T_1 and *should not* separate any pair of T_2.

- We prove that the EXACT STABLE BIPARTIZATION problem (Is there an independent set of size *exactly* k whose removal makes the graph bipartite?) is fixed-parameter tractable (FPT) answering an open question posed in 2001 by Díaz et al. [9]. We establish this result by proving that the STABLE BIPARTIZATION problem (Is there an independent set of size *at most* k whose removal makes the graph bipartite?) is FPT, answering an open question posed by Fernau [7].
- We show that the EDGE-INDUCED VERTEX CUT (Are there at most k edges such that the removal of their endpoints separates two given terminals s and t?) is FPT, answering an open problem posed in 2007 by Samer [7]. The motivation behind this problem is described in [27].

We believe that the above results nicely demonstrate the message of the paper. Slightly changing the definition of a well-understood cut problem usually makes the problem NP-hard and determining the parameterized complexity of such variants directly is by no means obvious. On the other hand, using our techniques, the fixed-parameter tractability of many such problems can be shown with very little effort. Let us mention (without proofs) three more variants that can be treated in a similar way: (1) separate s and t by the deletion of at most k edges and at most k vertices, (2) in a 2-colored graph, separate s and t by the deletion of at most k black and at most k white vertices, (3) in a k-colored graph, separate s and t by the deletion of one vertex from each color class.

As the examples above show, our method leads to the solution of several independent problems; it seems that the same combinatorial difficulty lies at the heart of these problems. Our technique manages to overcome this difficulty and it is expected to be of use for further problems of similar flavor. Note that while designing FPT-time algorithms for bounded-treewidth graphs and in particular the use of Courcelle's Theorem is a fairly standard technique, we use this technique for problems where there is no bound on the treewidth of the graph appearing in the input.

(Multiterminal) cut problems [19, 16, 13, 5] play a mysterious, and not yet fully understood, role in the fixed-parameter tractability of certain problems. Proving that BIPARTIZATION [25], DIRECTED FEEDBACK VERTEX SET [6], and ALMOST 2-SAT [23] are FPT answered longstanding open questions, and in each case the algorithm relies on a non-obvious use of separators. Furthermore, EDGE MULTICUT has been observed to be equivalent to FUZZY CLUSTER EDITING, a correlation clustering problem [3, 8, 1]. Thus aiming for a better understanding of separators in a parameterized setting seems to be a fruitful direction of research. Our results extend our understanding of separators by showing that various additional constraints can be accommodated. It is important to point out that our algorithm is very different from previous parameterized algorithms for separation problems [19, 16, 13, 5]. Those algorithms in the literature exploit certain nice properties of separators, and hence it seems impossible to generalize them for the problems we consider here. On the other hand, our approach is very robust and, as demonstrated by our examples, it is able to handle many variants.

The paper assumes the knowledge of the definition of treewidth and its algorithmic use, including Courcelle's Theorem (see the surveys [2, 14]).

2. Treewidth Reduction

The main combinatorial result of the paper is presented in this section. We start with some preliminary definitions. Two slightly different notions of separation will be used in the paper:

Definition 2.1. We say that a set S of vertices *separates* sets of vertices A and B if no component of $G \setminus S$ contains vertices from both $A \setminus S$ and $B \setminus S$. If s and t are two distinct vertices of G, then an $s - t$ *separator* is a set S of vertices disjoint from $\{s, t\}$ such that s and t are in different components of $G \setminus S$.

In particular, if S separates A and B, then $A \cap B \subseteq S$. Furthermore, given a set W of vertices, we say that a set S of vertices is a *balanced separator* of W if $|W \cap C| \leq |W|/2$ for every connected component C of $G \setminus S$. A k*-separator* is a separator S with $|S| = k$. The treewidth of a graph is closely connected with the existence of balanced separators:

Lemma 2.2 ([24], [12, Section 11.2])**.**

(1) *If $G(V, E)$ has treewidth greater than $3k$, then there is a set $W \subseteq V$ of size $2k + 1$ having no balanced k-separator.*

(2) *If $G(V, E)$ has treewidth at most k, then every $W \subseteq V$ has a balanced $(k + 1)$-separator.*

Note that the contrapositive of (1) in Lemma 2.2 says that if every set W of vertices has a balanced k-separator, then the treewidth is at most $3k$. This observation, and the following simple extension, will be convenient tools for showing that a certain graph has low treewidth.

Lemma 2.3. *Let G be a graph, C_1,..., C_r subsets of vertices, and let $C := \bigcup_{i=1}^{r} C_i$. Suppose that every $W_i \subseteq C_i$ has a balanced separator $S_i \subseteq C_i$ of size at most w. Then every $W \subseteq C$ has a balanced separator $S \subseteq C$ of size wr.*

If we are interested in separators of a graph G contained in a subset C of vertices, then each component of $G \setminus C$ (or the neighborhood of each component in C) can be replaced by a clique, since there is no way to disconnect these components with separators in C. The notion of torso and Proposition 2.5 formalize this concept.

Definition 2.4. Let G be a graph and $C \subseteq V(G)$. The graph $\mathrm{torso}(G, C)$ has vertex set C and vertices $a, b \in C$ are connected by an edge if $\{a, b\} \in E(G)$ or there is a path P in G connecting a and b whose internal vertices are not in C.

Proposition 2.5. *Let $C_1 \subseteq C_2$ be two subsets of vertices in G and let $a, b \in C_1$ be two vertices. A set $S \subseteq C_1$ separates a and b in $\mathrm{torso}(G, C_1)$ if and only if S separates these vertices in $\mathrm{torso}(G, C_2)$. In particular, by setting $C_2 = V(G)$, we get that $S \subseteq C_1$ separates a and b in $\mathrm{torso}(G, C_1)$ if and only if it separates them in G.*

Analogously to Lemma 2.3, we can show that if we have a treewidth bound on $\mathrm{torso}(G, C_i)$ for every i, then these bounds add up for the union of the C_i's.

Lemma 2.6. *Let G be a graph and C_1,..., C_r be subsets of $V(G)$ such that for every $1 \leq i \leq r$, the treewidth of $\mathrm{torso}(G, C_i)$ is at most w. Then the treewidth of $\mathrm{torso}(G, C)$ for $C := \bigcup_{i=1}^{r} C_i$ is at most $3r(w + 1)$.*

If the minimum size of an $s - t$ separator is ℓ, then the *excess* of an $s - t$ separator S is $|S| - \ell$ (which is always nonnegative). Note that if s and t are adjacent, then no $s - t$ separator exists, and in this case we say that the minimum size of an $s - t$ separator is ∞. The aim of this section is to show that, for every k, we can construct a set C' covering all the $s - t$ separators of size at most k such that torso(G, C') has treewidth bounded by a function of k. Equivalently, we can require that C' covers every $s - t$ separator of excess at most $e := k - \ell$, where ℓ is the minimum size of an $s - t$ separator.

If X is a set of vertices, we denote by $\delta(X)$ the set of those vertices in $V(G) \setminus X$ that are adjacent to at least one vertex of X. The following result is folklore; it can be proved by a simple application of the uncrossing technique (see the proof below) and it can be deduced also from the observations of [22] on the strongly connected components of the residual graph after solving a flow problem.

Lemma 2.7. *Let s, t be two vertices in graph G such that the minimum size of an $s - t$ separator is ℓ. Then there is a collection $\mathcal{X} = \{X_1, \dots, X_q\}$ of sets where $\{s\} \subseteq X_i \subseteq V(G) \setminus (\{t\} \cup \delta(\{t\}))$ ($1 \le i \le q$), such that*

(1) $X_1 \subset X_2 \subset \cdots \subset X_q$,
(2) $|\delta(X_i)| = \ell$ *for every* $1 \le i \le q$, *and*
(3) *every $s - t$ separator of size ℓ is a subset of* $\bigcup_{i=1}^{q} \delta(X_i)$.

Furthermore, such a collection $\mathcal{X}$ can be found in polynomial time.

Proof. Let $\mathcal{X} = \{X_1, \dots, X_q\}$ be a collection of sets such that (2) and (3) holds. Let us choose the collection such that q is the minimum possible, and among such collections, $\sum_{i=1}^{q} |X_i|^2$ is the maximum possible. We show that for every i, j, either $X_i \subset X_j$ or $X_j \subset X_i$ holds, thus the sets can be ordered such that (1) holds.

Suppose that neither $X_i \subset X_j$ nor $X_j \subset X_i$ holds for some i and j. We show that after replacing X_i and X_j in $\mathcal{X}$ with the two sets $X_i \cap X_j$ and $X_i \cup X_j$, properties (2) and (3) still hold, and the resulting collection $\mathcal{X}'$ contradicts the optimal choice of $\mathcal{X}$. The function δ is well-known to be submodular, i.e.,

$$|\delta(X_i)| + |\delta(X_j)| \ge |\delta(X_i \cap X_j)| + |\delta(X_i \cup X_j)|.$$

Both $\delta(X_i \cap X_j)$ and $\delta(X_i \cup X_j)$ are $s - t$ separators (because both $X_i \cap X_j$ and $X_i \cup X_j$ contain s) and hence have size at least k. The left hand side is 2ℓ, hence there is equality and $|\delta(X_i \cap X_j)| = |\delta(X_i \cup X_j)| = \ell$ follows. This means that property (2) holds after the replacement. Observe that $\delta(X_i \cap X_j) \cup \delta(X_i \cup X_j) \subseteq \delta(X_i) \cup \delta(X_j)$: any edge that leaves $X_i \cap X_j$ or $X_i \cup X_j$ leaves either X_i or X_j. We show that there is equality here, implying that property (3) remains true after the replacement. It is easy to see that $\delta(X_i \cap X_j) \cap \delta(X_i \cup X_j) \subseteq \delta(X_i) \cap \delta(X_j)$, hence we have

$$|\delta(X_i \cap X_j) \cup \delta(X_i \cup X_j)| = 2\ell - |\delta(X_i \cap X_j) \cap \delta(X_i \cup X_j)| \ge 2\ell - |\delta(X_i) \cap \delta(X_j)| = |\delta(X_i) \cup \delta(X_j)|,$$

showing the required equality.

If $X_i \cap X_j$ or $X_i \cup X_j$ was already present in $\mathcal{X}$, then the replacement decreases the size of the collection, contradicting the choice of $\mathcal{X}$. Otherwise, we have that $|X_i|^2 + |X_j|^2 < |X_i \cap X_j|^2 + |X_i \cup X_j|^2$ (to verify this, simply represent $|X_i|$ as $|X_i \cap X_j| + |X_i \setminus X_j|$, $|X_j|$ as $|X_i \cap X_j| + |X_j \setminus X_i|$, $|X_i \cup X_j|$ as $|X_i \cap X_j| + |X_i \setminus X_j| + |X_j \setminus X_i|$ and do direct calculation having in mind that both $|X_i \setminus X_j|$ and $|X_j \setminus X_i|$ are greater than 0), again contradicting the choice of $\mathcal{X}$. Thus an optimal collection $\mathcal{X}$ satisfies (1) as well.

To construct $\mathcal{X}$ in polynomial time, we proceed as follows. It is easy to check in polynomial time whether a vertex v is in a minimum $s - t$ separator, and if so to produce such a separator S_v.

Let X_v be the set of vertices reachable from s in $G \setminus S_v$. It is clear that X_v satisfies (2) and if we take the collection $\mathcal{X}$ of all such X_v's, then together they satisfy (3). If (1) is not satisfied, then we start doing the replacements as above. Each replacement either decreases the size of the collection or increases $\sum_{i=1}^{t} |X_i|^2$ (without increasing the collection size), thus the procedure terminates after a polynomial number of steps. ■

Lemma 2.7 shows that the union C of all minimum $s-t$ separators can be covered by a chain of minimum $s-t$ separators. It is not difficult to see that this chain can be used to define a tree decomposition (in fact, a path decomposition) of torso(G, C). This observation solves the problem for $e = 0$. For the general case, we use induction on e.

Lemma 2.8. *Let s, t be two vertices of graph G and let ℓ be the minimum size of an $s-t$ separator. For some $e \geq 0$, let C be the union of all minimal $s-t$ separators having* excess *at most e (i.e. of size at most $k = \ell + e$). Then, for some constant d, there is an $O(f(\ell, e) \cdot |V(G)|^d)$ time algorithm that returns a set $C' \supseteq C \cup \{s, t\}$ such that the treewidth of* torso(G, C') *is at most $g(\ell, e)$, where functions f and g depend only on ℓ and e .*

Proof. We prove the lemma by induction on e. Consider the collection $\mathcal{X}$ of Lemma 2.7 and define $S_i := \delta(X_i)$ for $1 \leq i \leq q$. For the sake of uniformity, we define $X_0 := \emptyset$, $X_{q+1} := V(G) \setminus \{t\}$, $S_0 := \{s\}$, $S_{q+1} := \{t\}$. For $1 \leq i \leq q+1$, let $L_i := X_i \setminus (X_{i-1} \cup S_{i-1})$. Also, for $1 \leq i \leq q+1$ and two disjoint *non-empty* subsets A, B of $S_i \cup S_{i-1}$, we define $G_{i,A,B}$ to be the graph obtained from $G[L_i \cup A \cup B]$ by contracting the set A to a vertex a and the set B to a vertex b. Taking into account that if C includes a vertex of some L_i then $e > 0$, we prove the key observation that makes it possible to use induction.

Claim 2.9. If a vertex $v \in L_i$ is in C, then there are disjoint non-empty subsets A, B of $S_i \cup S_{i-1}$ such that v is part of a minimal $a-b$ separator K_2 in $G_{i,A,B}$ of size at most k (recall that $k = \ell + e$) and excess at most $e-1$.

Proof. By definition of C, there is a minimal $s-t$ separator K of size at most k that contains v. Let $K_1 := K \setminus L_i$ and $K_2 := K \cap L_i$. Partition $(S_i \cup S_{i-1}) \setminus K$ into the set A of vertices reachable from s in $G \setminus K$ and the set B of vertices non-reachable from s in $G \setminus K$. Let us observe that both A and B are non-empty. Indeed, due to the minimality of K, G has a path P from s to t such that $V(P) \cap K = \{v\}$. By selection of v, S_{i-1} separates v from s and S_i separates v from t. Therefore, at least one vertex u of S_{i-1} occurs in P before v and at least one vertex w of S_i occurs in P after v. The prefix of P ending at u and the suffix of P starting at w are both subpaths in $G \setminus K$. It follows that u is reachable from s in $G \setminus K$, i.e. belongs to A and that w is reachable from t in $G \setminus K$, hence non-reachable from s and thus belongs to B.

To see that K_2 is an $a-b$ separator in $G_{i,A,B}$, suppose that there is a path P connecting a and b in $G_{i,A,B}$ avoiding K_2. Then there is a corresponding path P' in G connecting a vertex of A and a vertex of B. Path P' is disjoint from K_1 (since it contains vertices of L_i and $(S_i \cup S_{i-1}) \setminus K$ only) and from K_2 (by construction). Thus a vertex of B is reachable from s in $G \setminus K$, a contradiction.

To see that K_2 is a minimal $a-b$ separator, suppose that there is a vertex $u \in K_2$ such that $K_2 \setminus \{u\}$ is also an $a-b$ separator in $G_{i,A,B}$. Since K is minimal, there is an $s-t$ path P in $G \setminus (K \setminus u)$, which has to pass through u. Arguing as when we proved that A and B are non-empty, we observe that P includes vertices of both A and B, hence we can consider a minimal subpath P' of P between a vertex $a' \in A$ and a vertex $b' \in B$. We claim that all the internal vertices of P' belong to L_i. Indeed, due to the minimality of P', an internal vertex of P' can belong either to L_i or to $V(G) \setminus (K_1 \cup L_i \cup S_{i-1} \cup S_i)$. If all the internal vertices of P' are from the latter set then there is a path from a' to b' in $G \setminus (K_1 \cup L_i)$ and hence in $G \setminus (K_1 \cup K_2)$ in contradiction to

$b' \in B$. If P' contains internal vertices of both sets then G has an edge $\{u, w\}$ where $u \in L_i$ while $w \in V(G) \setminus (K_1 \cup L_i \cup S_{i-1} \cup S_i)$. But this is impossible since $S_{i-1} \cup S_i$ separates L_i from the rest of the graph. Thus it follows that indeed all the internal vertices of P' belong to L_i. Consequently, P' corresponds to a path in $G_{i,A,B}$ from a to b that avoids $K_2 \setminus u$, a contradiction that proves the minimality of K_2.

Finally, we show that K_2 has excess at most $e - 1$. Let K_2' be a minimum $a - b$ separator in $G_{i,A,B}$. Observe that $K_1 \cup K_2'$ is an $s - t$ separator in G. Indeed, consider a path P from s to t in $G \setminus (K_1 \cup K_2')$. It necessarily contains a vertex $u \in K_2$, hence arguing as in the previous paragraph we notice that P includes vertices of both A and B. Considering a minimal subpath P' of P between a vertex $a' \in A$ and $b' \in B$ we observe, analogously to the previous paragraph that all the internal vertices of this path belong to L_i. Hence this path corresponds to a path between a and b in $G_{i,A,B}$. It follows that P', and hence P, includes a vertex of K_2', a contradiction showing that $K_1 \cup K_2'$ is indeed an $s - t$ separator in G. Due to the minimality of K_2, $K_2' \neq \emptyset$. Thus $K_1 \cup K_2'$ contains at least one vertex from L_i, implying that $K_1 \cup K_2'$ is not a minimum $s - t$ separator in G. Thus $|K_2| - |K_2'| = (|K_1| + |K_2|) - (|K_1| + |K_2'|) < k - \ell = e$, as required. This completes the proof of Claim 2.9. ■

Now we define C'. Let $C_0 := \bigcup_{i=0}^{q+1} S_i$. For $e = 0$, $C' = C_0$. Assume that $e > 0$. For $1 \leq i \leq q + 1$ and disjoint non-empty subsets A, B of $S_i \cup S_{i-1}$. Let $C_{i,A,B}$ be such a superset of the union of all minimal $a - b$ separators of $G_{i,A,B}$ of size most k and excess at most $e - 1$ that $C_{i,A,B} \cup \{a, b\}$ satisfies the induction assumption with respect to $G_{i,A,B}$ (if the minimum size of an $a - b$ separator of $G_{i,A,B}$ is greater than k then we set $C_{i,A,B} = \emptyset$). We define C' as the union of C_0 and all sets $C_{i,A,B}$ as above. Observe that C' is defined correctly in the sense that any vertex v participating in an $s - t$ minimal separator of size at most k indeed belongs to C'. For $e = 0$, the correctness of C' follows from the definition of sets S_i. For $e > 0$, the correctness follows from the above Claim if we take into account that since $\bigcup_{i=1}^{q+1} L_i \cup C_0 = V(G)$, v belongs to some L_i.

We shall show that the treewidth of torso(G, C') is at most $g(\ell, e)$, a function recursively defined as follows: $g(\ell, 0) := 6\ell$ and $g(\ell, e) := 3 \cdot (2\ell + 3^{2\ell} \cdot (g(\ell, e - 1) + 1))$ for $e > 0$. We do this by showing that in graph G, every set $W \subseteq C'$ has a balanced separator of size at most 2ℓ (for $e = 0$) and at most $2\ell + 3^{2\ell} \cdot (g(\ell, e - 1) + 1)$ (for $e > 0$). By Proposition 2.5, this will imply that in torso(G, C'), W has a balanced separator with the same upper bound. By Lemma 2.2(1), the desired upper bound on the treewidth will immediately follow.

Let $W \subseteq C'$ be an arbitrary set. Let $1 \leq i \leq q + 1$ be the smallest value such that $|W \cap X_i| \geq |W|/2$. Consider the separator $S_i \cup S_{i-1}$ (whose size is at most 2ℓ). In $G \setminus (S_i \cup S_{i-1})$, the sets X_{i-1}, L_i, and $V(G) \setminus (S_i \cup S_{i-1} \cup X_{i-1} \cup L_i)$ are pairwise separated from each other. By the selection of i, the first and the third sets do not contain more than half of W. If $e = 0$, then C' is disjoint from L_i, hence the treewidth upper bound follows for $e = 0$. We assume that $e > 0$ and, using the induction assumption, will show that $W \cap L_i$ has a balanced separator S of size at most $3^{2\ell} \cdot (g(\ell, e - 1) + 1)$. This will immediately imply that $S \cup S_i \cup S_{i-1}$ is a balanced separator of W of size at most $2\ell + 3^{2\ell} \cdot (g(\ell, e - 1) + 1)$, which, in turn, will imply the desired upper bound on the treewidth of torso(G, C').

By the induction assumption, the treewidth of torso$(G_{i,A,B}, C_{i,A,B})$ is at most $g(\ell, e - 1)$ for any pair of disjoint subsets A, B of $S_i \cup S_{i-1}$ such that $G_{i,A,B}$ has an $a - b$ separator of size at most k. By the combination of Lemma 2.2(2) and Proposition 2.5, graph G has a balanced separator of size at most $g(\ell, e - 1) + 1$ for any set $W_{i,A,B} \subseteq C_{i,A,B}$. Let C^* be the union of $C_{i,A,B}$ for all such A and B. Taking into account that the number of choices of A and B is at most $3^{2\ell}$, for any

$W^* \subseteq C^*$, G has a balanced separator of size at most $3^{2\ell} \cdot (g(\ell, e-1)+1)$ according to Lemma 2.3. By definition of C', $W \cap L_i \subseteq C^*$, hence the existence of the desired separator S follows.

We conclude the proof by showing that the above set C' can be constructed in time $O(f(\ell, e) \cdot |V(G)|^d)$. In particular, we present an algorithm whose running time is $O(f(\ell, e) \cdot (|V(G)| - 2)^d)$ (we assume that G has more than 2 vertices), where $f(\ell, e)$ is recursively defined as follows: $f(\ell, 0) = 1$ and $f(\ell, e) = f(\ell, e-1) \cdot 3^{2\ell} + 1$ for $e > 0$.

The set X_i can be computed as shown in the proof of Lemma 2.7. Then the set S_i can be obtained as in the first paragraph of the proof of the present lemma. Their union results in C_0 which is C' for $e = 0$. Thus for $e = 0$, C' can be computed in time $O(|V(G)| - 2)^d)$ (instead of considering s and t, we may consider their sets of neighbors). Since the computation involves computing a minimum cut, we may assume that $d > 1$. Now assume that $e > 0$. For each i such that $1 \leq i \leq q+1$ and $|L_i| > 0$, we explore all possible disjoint subsets A and B of $S_i \cup S_{i-1}$. For the given choice, we check if the size of a minimum $a - b$ separator of $G_{i,A,B}$ is at most k (observe that it can be done in $O(|L_i|^d))$ and if yes, compute the set $C_{i,A,B}$. By the induction assumption, the computation takes $O(f(\ell, e-1) \cdot |L_i|^d)$. So, exploring all possible choices of A and B takes $O(f(\ell, e-1) \cdot 3^{2\ell} \cdot |L_i|^d)$. The overall complexity of computing C' is

$$O((|V(G)| - 2)^d + f(\ell, e-1) \cdot 3^{2\ell} \cdot \sum_{i=1}^{q+1} |L_i|^d).$$

Since all L_i are disjoint and $\bigcup_{i=1}^{q+1} L_i \subseteq V(G) \backslash \{s, t\}$, $\sum_{i=1}^{q+1} |L_i| \leq |V(G)| - 2$, hence $\sum_{i=1}^{q+1} (|L_i|)^d \leq (|V(G)| - 2)^d$. Taking into account the recursive expression for $f(\ell, e)$, the desired runtime follows. ■

Remark 2.10. The recursion $g(\ell, e) := 3 \cdot (2\ell + 3^{2\ell} \cdot g(\ell, e-1))$ implies that $g(\ell, e)$ is $2^{O(e\ell)}$, i.e., the treewidth bound is exponential in ℓ and e. It is an obvious question whether it is possible to improve this dependence to polynomial. However, a simple example (graph G is the n-dimensional hypercube, $k = (n-1)n$, s and t are opposite vertices) shows that the function $g(\ell, e)$ has to be exponential. The size of the minimum $s - t$ separator is $\ell := n$. We claim that every vertex v of the hypercube (other than s and t) is part of a minimal $s - t$ separator of size at most $n(n-1)$. To see this, let P be a shortest path connecting s and v. Let $P' = P - v$ be the subpath of P connecting s with a neighbor v' of v. Let S be the neighborhood of P'; clearly S is an $s - t$ separator and $v \in S$. However, $S \setminus v$ is not an $s - t$ separator: the path P is not blocked by $S \setminus v$ as $S \setminus v$ does not contain any vertex farther from s than v. Since P' has at most $n - 1$ vertices and every vertex has degree n, we have $|S| \leq n(n-1)$. Thus v (and every other vertex) is part of a minimal separator of size at most $n(n-1)$. Hence if we set $\ell := n$ and $e := n(n-1)$, then C contains every vertex of the hypercube. The treewidth of an n-dimensional hypercube is $\Omega(2^n/\sqrt{n})$ [4], which is also a lower bound on $g(\ell, e)$.

The following theorem states our main combinatorial tool in a form that will be very convenient to use.

Theorem 2.11 (The Treewidth Reduction Theorem). *Let G be a graph, $S \subseteq V(G)$, and let k be an integer. Let C be the set of all vertices of G participating in a minimal $s - t$ cut of size at most k for some $s, t \in S$. Then there is an* FPT *algorithm, parameterized by k and $|S|$, that computes a graph G^* having the following properties:*

(1) $C \cup S \subseteq V(G^*)$

(2) *For every $s, t \in S$, a set $K \subseteq V(G^*)$ with $|K| \leq k$ is a minimal $s - t$ separator of G^* if and only if $K \subseteq C \cup S$ and K is a minimal $s - t$ separator of G.*

(3) *The treewidth of G^* is at most $h(k, |S|)$ for some function h.*
(4) *For any $K \subseteq C$, $G^*[K]$ is isomorphic to $G[K]$.*

Proof. For every $s, t \in S$ that can be separated by the removal of at most k vertices, the algorithm of Lemma 2.8 computes a set $C'_{s,t}$ containing all the minimal $s - t$ separators of size at most k. By Lemma 2.6, if C' is the union of these at most $\binom{|S|}{2}$ sets, then $G' = \text{torso}(G, C')$ has treewidth bounded by a function of k and $|S|$. Note that G' satisfies all the requirements of the theorem except the last one: two vertices of C' non-adjacent in G may become adjacent in G' (see Definition 2.4). To fix this problem we subdivide each edge $\{u, v\}$ of G' such that $\{u, v\} \notin E(G)$ into two edges with a vertex between them, and, to avoid selecting this vertex into a cut, we split it into $k+1$ copies. In other words, for each edge $\{u, v\} \in E(G') \setminus E(G)$ we introduce $k+1$ new vertices $w_1, \dots, w_{k+1}$ and replace $\{u, v\}$ by the set of edges $\{\{u, w_1\}, \dots, \{u, w_{k+1}\}, \{w_1, v\}, \dots, \{w_{k+1}, v\}\}$. Let G^* be the resulting graph. It is not hard to check that G^* satisfies all the properties of the present theorem. ■

Remark 2.12. The treewidth of G^* may be larger than the treewidth of G. We use the phrase "treewidth reduction" in the sense that the treewidth of G^* is bounded by a function of k and $|S|$, while the treewidth of G is unbounded.

3. Constrained Separation Problems

Let $\mathcal{G}$ be a class of graphs. Given a graph G, vertices s and t, and parameter k, the $\mathcal{G}$-MINCUT problem asks if G has an $s - t$ separator C of size at most k such that $G[C] \in \mathcal{G}$. The following theorem is the central result of this section.

Theorem 3.1. *Assume that $\mathcal{G}$ is* decidable *and* hereditary *(i.e. whenever $G \in \mathcal{G}$ then for any $V' \subseteq V$, $G[V'] \in \mathcal{G}$). Then the $\mathcal{G}$-MINCUT problem is FPT.*

Proof. (Sketch) Let G^* be a graph satisfying the requirements of Theorem 2.11 for $S = \{s, t\}$. According to Theorem 2.11, G^* can be computed in FPT time. We claim that (G, s, t, k) is a 'YES' instance of the $\mathcal{G}$-MINCUT problem if and only if (G^*, s, t, k) is a 'YES' instance of this problem. Indeed, let K be an $s - t$ separator in G such that $|K| \leq k$ and $G(K) \in \mathcal{G}$. Since $\mathcal{G}$ is hereditary, we may assume that K is minimal (otherwise we may consider a minimal subset of K separating s from t). By the second and fourth properties of G^* (see Theorem 2.11), K separates s from t in G^* and $G^*[K] \in \mathcal{G}$. The opposite direction can be proved similarly.

Thus we have established an FPT-time reduction from an instance of the $\mathcal{G}$-MINCUT problem to another instance of this problem where the treewidth is bounded by a function of parameter k. Now, let $G_1 = (V(G^*), E(G^*), ST)$ be a labeled graph where $ST = \{s, t\}$. We present an algorithm for constructing a monadic second-order (MSO) formula φ whose atomic predicates (besides equality) are $E(x_1, x_2)$ (showing that x_1 and x_2 are adjacent in G^*) and predicates of the form $X(v)$ (showing that v is contained in $X \subseteq V$), whose size is bounded by a function of k, and $G_1 \models \varphi$ if and only if (G^*, s, t, k) is a 'YES' instance of the $\mathcal{G}$-MINCUT problem. According to a restricted version of the well-known Courcelle's Theorem (see the survey article of Grohe [14], Remarks 3.19[1] and 3.20), it will follow that the $\mathcal{G}$-MINCUT problem is FPT. The part of φ describing the separation of s and t is based on the ideas from [13].

[1] Although the branchwidth of G_1 appears in the parameter, it can be replaced by the treewidth of G_1 since the former is bounded by a function of k if and only if the latter is [26].

We construct the formula φ as

$$\varphi = \exists C(\text{AtMost}_k(C) \wedge \text{Separates}(C) \wedge \text{Induces}_{\mathcal{G}}(C)),$$

where $\text{AtMost}_k(C)$ is true if and only if $|C| \leq k$, $\text{Separates}(C)$ is true if and only if C separates the vertices of ST in G^*, and $\text{Induces}_{\mathcal{G}}(C)$ is true if and only C induces a graph of $\mathcal{G}$.

In particular, $\text{AtMost}_k(C)$ states that C does not have $k+1$ mutually non-equal elements: this can be implemented as

$$\forall c_1, \ldots, \forall c_{k+1} \bigvee_{1 \leq i,j \leq k+1} (c_i = c_j).$$

Formula $\text{Separates}(C)$ is a slightly modified formula $\text{uvmc}(X)$ from [13], that looks as follows:

$$\forall s \forall t \forall Z \big(ST(s) \wedge ST(t) \wedge \neg(s = t) \wedge \neg C(s) \wedge \neg C(t) \wedge \text{Connects}(Z, s, t)\big) \rightarrow \big(\exists v(C(v) \wedge Z(v))\big)\big),$$

where $\text{Connects}(Z, s, t)$ is true if and only if in the modeling graph there is a path from s and t all vertices of which belong to Z. For the definition of the predicate Connects, see Definition 3.1 in [13].

To construct $\text{Induces}_{\mathcal{G}}(C)$, we explore all possible graphs having at most k vertices and for each of these graphs we check whether it belongs to $\mathcal{G}$. Since the number of graphs being explored depends on k and $\mathcal{G}$ is a decidable class, in FPT time we can compile the set $\{G'_1, \ldots, G'_r\}$ of all graphs of at most k vertices that belong to $\mathcal{G}$. Let $k_1, \ldots k_r$ be the respective numbers of vertices of $G'_1, \ldots G'_r$. Then $\text{Induces}_{\mathcal{G}}(C) = \text{Induces}_1(C) \vee \cdots \vee \text{Induces}_r(C)$, where $\text{Induces}_i(C)$ states that C induces G'_i. To define Induces_i, let $v_1, \ldots, v_{k_i}$ be the set of vertices of G'_i and define $\text{Adj}_i(c_1, \ldots, c_{k_i})$ as the conjunction of all $E(c_x, c_y)$ such that v_x and v_y are adjacent in G'_i and of all $\neg E(c_x, c_y)$ such that v_x and v_y are not adjacent in G'_i. Then

$$\text{Induces}_i(C) = \text{AtMost}_{k_i}(C) \wedge \exists c_1 \ldots \exists c_{k_i} \Big(\bigwedge_{1 \leq j \leq k_i} C(c_j) \wedge \bigwedge_{1 \leq x,y \leq k_i} c_x \neq c_y \wedge \text{Adj}_i(c_1, \ldots, c_{k_i}) \Big).$$

It is not hard to verify that indeed $G_1 \models \varphi$ if and only if (G^*, s, t, k) is a 'YES' instance of the $\mathcal{G}$-MINCUT problem. ■

In particular, let $\mathcal{G}^0$ be the class of all graphs without edges. Then $\mathcal{G}^0$-MINCUT is the MINIMUM STABLE $s-t$ CUT problem whose fixed-parameter tractability has been posed as an open question by Kanj [17]. Clearly, $\mathcal{G}^0$ is hereditary and hence the $\mathcal{G}^0$-MINCUT is FPT.

Theorem 3.1 can be used to decide if there is an $s-t$ separator of size *at most* k having a certain property, but cannot be used if we are looking for $s-t$ separators of size *exactly* k. We show (with a very easy argument) that some of these problems actually become hard if the size is required to be exactly k. Let graph G' be obtained from graph G by introducing two isolated vertices s and t. Now there is an independent set of size exactly k that is an $s-t$ separator in G' if and only if there is an independent set of size k in G, implying that finding such a separator is W[1]-hard.

Theorem 3.2. *It is* W[1]*-hard to decide if G has an $s-t$ separator that is an independent set of size exactly k.*

Samer and Szeider [27] introduced the notion of *edge-induced vertex-cut* and the corresponding computational problem: given a graph G and two vertices s and t, the task is to decide if there are k edges such that deleting the *endpoints* of these edges separates s and t. It remained an open question in [27] whether this problem is FPT. Samer reposted this problem as an open question in [7]. Using Theorem 3.1, we answer this question positively. For this purpose, we introduce $\mathcal{G}_k$, the class of graphs where the number of vertices minus the size of the maximum matching is at most k, observe that this class is hereditary, and show that (G, s, t, k) is a 'YES'-instance of the *edge-induced vertex-cut* problem if and only if $(G, s, t, 2k)$ is a 'YES' instance of the $\mathcal{G}_k$-mincut problem. Then we apply Theorem 3.1 to get the following corollary.

Corollary 3.3. *The* EDGE-INDUCED VERTEX-CUT *problem is* FPT.

MULTICUT is the generalization of MINCUT where, instead of s and t, the input contains a set (s_1, t_1), ..., (s_ℓ, t_ℓ) of terminal pairs. The task is to find a set S of at most k nonterminal vertices that separate s_i and t_i for every $1 \leq i \leq \ell$. MULTICUT is known to be FPT [19, 28] parameterized by k and ℓ. In the $\mathcal{G}$-MULTICUT problem, we additionally require that S induces a graph from $\mathcal{G}$. It is not difficult to generalize Theorem 3.1 for $\mathcal{G}$-MULTICUT: all we need to do is to change the construction of φ such that it requires the separation of each pair (s_i, t_i). We state this here in an even more general form. In the $\mathcal{G}$-MULTICUT-UNCUT problem the input contains an additional integer $\ell' \leq \ell$, and we change the problem by requiring for every $\ell' \leq i \leq \ell$ that S *does not* separate s_i and t_i.

Theorem 3.4. *If $\mathcal{G}$ is* decidable *and* hereditary, *then $\mathcal{G}$-*MULTICUT-UNCUT *is* FPT *parameterized by k and ℓ.*

Theorem 3.4 helps clarify a theoretical issue. In Section 2, we defined C as the set of all vertices appearing in minimal $s - t$ separators of size at most k. There is no obvious way of finding this set in FPT-time and Lemma 2.6 produces only a superset C' of C. However, Theorem 3.4 can be used to find C: a vertex v is in C if and only if there is a set S of size at most $k - 1$ and two neighbors v_1, v_2 of v such that S separates s and t in $G \setminus v$, but S does not separate s from v_1 and t from v_2 in $G \setminus v$ (including the possibility that $v_1 = s$ or $v_2 = t$).

4. Constrained Bipartization Problems

Reed et al. [25] solved a longstanding open question by proving the fixed-parameter tractability of the BIPARTIZATION problem: given a graph G and an integer k, find a set S of at most k vertices such that $G \setminus S$ is bipartite (see also [18] for a somewhat simpler presentation of the algorithm). In fact, they showed that the BIPARTIZATION problem can be solved by at most 3^k applications of a procedure solving MINCUT. The key result that allows to transform BIPARTIZATION to a separation problem is the following lemma.

Lemma 4.1. *Let G be a bipartite graph and let (B', W') be a 2-coloring of the vertices. Let B and W be two subsets of $V(G)$. Then for any S, $G \setminus S$ has a 2-coloring where $B \setminus S$ is black and $W \setminus S$ is white if and only if S separates $X := (B \cap B') \cup (W \cap W')$ and $Y := (B \cap W') \cup (W \cap B')$.*

In this section we consider the $\mathcal{G}$-BIPARTIZATION problem: a generalization of the BIPARTIZATION problem where, in addition to $G \setminus S$ being bipartite, it is also required that S induces a graph belonging to a class $\mathcal{G}$.

Theorem 4.2. *$\mathcal{G}$-*BIPARTIZATION *is* FPT *if $\mathcal{G}$ is hereditary and decidable.*

Proof. Using the algorithm of [25], we first try to find a set S_0 of size at most k such that $G \setminus S_0$ is bipartite. If no such set exists, then clearly there is no set S satisfying the requirements. Otherwise, we branch in $3^{|S_0|}$ directions: each vertex of S_0 is removed or colored black or colored white. For a particular branch, let $R = \{v_1, \ldots, v_r\}$ be the vertices of S_0 to be removed and let B_0 (resp., W_0) be the vertices of S_0 having color black (resp., white) in a 2-coloring of the resulting bipartite graph. Let us call a set S such that $S \cap S_0 = R$, and $G \setminus S$ is bipartite and having a 2-coloring where

B_0 and W_0 are colored black and white, respectively, a set *compatible* with (R, B_0, W_0). Clearly, (G, k) is a 'YES' instance of the $\mathcal{G}$-BIPARTIZATION problem if and only if for at least one branch corresponding to partition (R, B_0, W_0) of S_0, there is a set compatible with (R, B_0, W_0) having size at most k and such that $G[S] \in \mathcal{G}$. Clearly, we need to check only those branches where $G[B_0]$ and $G[W_0]$ are both independent sets.

We transform the problem of finding a set compatible with (R, B_0, W_0) into a separation problem. Let (B', W') be a 2-coloring of $G \setminus S_0$. Let $B = N(W_0) \setminus S_0$ and $W = N(B_0) \setminus S_0$. Let us define X and Y as in Lemma 4.1, i.e., $X := (B \cap B') \cup (W \cap W')$, and $Y := (B \cap W') \cup (W \cap B')$. We construct a graph G' that is obtained from G by deleting the set $B_0 \cup W_0$, adding a new vertex s adjacent to $X \cup R$, and adding a new vertex t adjacent with $Y \cup R$. Note that every $s-t$ separator in G' contains R. By Lemma 4.1, a set S is compatible with (R, B_0, W_0) if and only if S is an $s-t$ separator in G'. Thus what we have to decide is whether there is an $s-t$ separator S of size at most k such that $G'[S] = G[S]$ is in $\mathcal{G}$. That is, we have to solve the $\mathcal{G}$-MINCUT instance (G', s, t, k). The fixed-parameter tractability of the $\mathcal{G}$-BIPARTIZATION problem now immediately follows from Theorem 3.1. ∎

Theorem 4.2 immediately implies that the STABLE BIPARTIZATION problem is FPT: just set $\mathcal{G}$ to be the class of all graphs without edges. This answers an open question of Fernau [7]. Next, we show that the EXACT STABLE BIPARTIZATION problem is FPT, answering a question posed by Díaz et al. [9]. This result may seem surprising because the corresponding exact separation problem is W[1]-hard by Theorem 3.2 and hence the approach of Theorem 4.2 is unlikely to work. Instead, we argue that under appropriate conditions, any solution of size at most k can be extended to an independent set of size exactly k.

Theorem 4.3. *Given a graph G and an integer k, deciding whether G can be made bipartite by the deletion of an independent set of size exactly k is fixed-parameter tractable.*

Proof. (Sketch) It is more convenient to consider an annotated version of the problem where the independent set being deleted has to be a subset of a set $D \subseteq V(G)$ given as part of the input. Without the annotation, D is initially set to $V(G)$. If G is not bipartite, then the algorithm starts by finding an odd cycle C of minimum length (which can be done in polynomial time). It is not difficult to see that the minimality of C implies that either C is a triangle or C is chordless. Moreover, in the latter case, every vertex not in C is adjacent to at most 2 vertices of the cycle.

If $|V(C) \cap D| = 0$, then clearly no subset of D is a solution. If $1 \leq |V(C) \cap D| \leq 3k+1$, then we branch on the selection of each vertex $v \in V(C) \cap D$ into the set S of vertices being removed and apply the algorithm recursively with the parameter k being decreased by 1 and the set D being updated by the removal of v and $N(v) \cap D$. If $|V(C) \cap D| > 3k + 1$, then we apply the approach of Theorem 4.2 to find an independent set $S \subseteq D$ of size at most k whose removal makes the graph bipartite, and then argue that S can be extended to an independent set of size exactly k. To ensure that $S \subseteq D$, we may, for example split all vertices $v \in V(G) \setminus D$ into $k + 1$ independent copies with the same neighborhood as v. If $|S| = k$, we are done. Otherwise, $|S| = k' < k$. In this case we observe that by the minimality of C, each vertex of S (either in C or outside C) forbids the selection of at most 3 vertices of $V(C) \cap D$ including itself. Thus the number of vertices of $V(C) \cap D$ allowed for selection is at least $3k+1-3k' = 3(k-k')+1$. Since the cycle is chordless, we can select $k - k'$ independent vertices among them and thus complement S to be of size exactly k.

The above algorithm has a number of stopping conditions, the only non-trivial of them occurs if G is bipartite but $k > 0$. In this case we check if $G[D]$ has k independent vertices, which can be done in a polynomial time. ∎

Acknowledgements

The research of Dániel Marx was supported by ERC Advanced Grant DMMCA. The research of Barry O'Sullivan and Igor Razgon was supported by Science Foundation Ireland through Grant 05/IN/I886. We would like to thank the anonymous referees for spotting a number of minor mistakes in the preliminary version of this paper. Fixing those mistakes in the camera-ready version allowed us to significantly improve its quality.

References

[1] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[2] H. L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *WG*, pages 1–14, 2006.

[3] H. L. Bodlaender, M. R. Fellows, P. Heggernes, F. Mancini, C. Papadopoulos, and F. A. Rosamond. Clustering with partial information. In *MFCS*, pages 144–155, 2008.

[4] L. S. Chandran and T. Kavitha. The treewidth and pathwidth of hypercubes. *Discrete Math.*, 306(3):359–365, 2006.

[5] J. Chen, Y. Liu, and S. Lu. An improved parameterized algorithm for the minimum node multiway cut problem. In *WADS*, pages 495–506, 2007.

[6] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008.

[7] E. Demaine, G. Z. Gutin, D. Marx, and U. Stege. Seminar 07281 Open problems. In *Structure Theory and FPT Algorithmics for Graphs, Digraphs and Hypergraphs*, 2007.

[8] E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2-3):172–187, 2006.

[9] J. Díaz, M. Serna, and D. M. Thilikos. (H, C, K)-coloring: fast, easy, and hard cases. In *MFCS 2001*, pages 304–315, 2001.

[10] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, New York, 1999.

[11] U. Feige and M. Mahdian. Finding small balanced separators. In *STOC 2006*, pages 375–384. ACM, 2006.

[12] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[13] G. Gottlob and S. T. Lee. A logical approach to multicut problems. *Inform. Process. Lett.*, 103(4):136–141, 2007.

[14] M. Grohe. Logic, graphs, and algorithms. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata- History and Perspectives*. Amsterdam University Press, 2007.

[15] S. Guillemot. FPT algorithms for path-transversals and cycle-transversals problems in graphs. In *IWPEC*, pages 129–140, 2008.

[16] J. Guo, F. Hüffner, E. Kenar, R. Niedermeier, and J. Uhlmann. Complexity and exact algorithms for vertex multicut in interval and bounded treewidth graphs. *European Journal of Operational Research*, 186(2):542–553, 2008.

[17] I. Kanj. Open problem session of Dagstuhl seminar 08431, 2008.

[18] D. Lokshtanov, S. Saurabh, and S. Sikdar. Simpler parameterized algorithm for OCT. In *IWOCA 2009*, pages 380–384, 2009.

[19] D. Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

[20] D. Marx and I. Razgon. Constant ratio fixed-parameter approximation of the edge multicut problem. In *ESA 2009*, pages 647–658, 2009.

[21] R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

[22] J.-C. Picard and M. Queyranne. On the structure of all minimum cuts in a network and applications. *Math. Programming Stud.*, (13):8–16, 1980. Combinatorial optimization, II (Proc. Conf., Univ. East Anglia, Norwich, 1979).

[23] I. Razgon and B. O'Sullivan. Almost 2-SAT is fixed-parameter tractable. In *ICALP (1)*, pages 551–562, 2008.

[24] B. Reed. Tree width and tangles: A new connectivity measure and some applications. In R. Bailey, editor, *Surveys in Combinatorics*, volume 241 of *LMS Lecture Note Series*, pages 87–162. Cambridge University Press, 1997.

[25] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

[26] N. Robertson and P. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991.

[27] M. Samer and S. Szeider. Complexity and applications of edge-induced vertex-cuts. *CoRR*, abs/cs/0607109, 2006.

[28] M. Xiao. Algorithms for multiterminal cuts. In *CSR*, pages 314–325, 2008.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 573-584
www.stacs-conf.org

ONLINE CORRELATION CLUSTERING

CLAIRE MATHIEU [1] AND OCAN SANKUR [1,2] AND WARREN SCHUDY [1]

[1] Department of Computer Science, Brown University, 115 Waterman Street, Providence, RI 02912

[2] Ecole Normale Supérieure, 45, rue d'Ulm, 75005 Paris, France

Abstract. We study the online clustering problem where data items arrive in an online fashion. The algorithm maintains a clustering of data items into similarity classes. Upon arrival of v, the relation between v and previously arrived items is revealed, so that for each u we are told whether v is similar to u. The algorithm can create a new cluster for v and merge existing clusters.

When the objective is to minimize disagreements between the clustering and the input, we prove that a natural greedy algorithm is O(n)-competitive, and this is optimal.

When the objective is to maximize agreements between the clustering and the input, we prove that the greedy algorithm is .5-competitive; that no online algorithm can be better than .834-competitive; we prove that it is possible to get better than 1/2, by exhibiting a randomized algorithm with competitive ratio .5+c for a small positive fixed constant c.

1. Introduction

We study online correlation clustering. In correlation clustering [2, 15], the input is a complete graph whose edges are labeled either *positive*, meaning similar, or *negative*, meaning dissimilar. The goal is to produce a clustering that agrees as much as possible with the edge labels. More precisely, the output is a clustering that maximizes the number of agreements, *i.e.*, the sum of positive edges within clusters and the negative edges between clusters. Equivalently, this clustering minimizes the disagreements. This has applications in information retrieval, e.g. [8, 10].

In the online setting, vertices arrive one at a time and the total number of vertices is unknown to the algorithm *a priori*. Upon the arrival of a vertex, the labels of the edges that connect this new vertex to the previously discovered vertices are revealed. The algorithm updates the clustering while preserving the clusters already identified (it is not permitted to split any pre-existing cluster). Motivated by information retrieval applications, this online model was proposed by Charikar, Chekuri, Feder and Motwani [5] (for another clustering problem). As in [5], our algorithms maintain *Hierarchical Agglomerative Clusterings* at all times; this is well suited for the applications of interest.

1998 ACM Subject Classification: F.2.2 Nonnumerical Algorithms and Problems.
Key words and phrases: correlation clustering, online algorithms.
Part of this work was funded by NSF grant CCF 0728816.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE
27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2486

The problem of correlation clustering was introduced by Ben-Dor et al. [3] to cluster gene expression patterns. Unfortunately, it was shown that even the offline version of correlation clustering is NP-hard [15, 2]. The following are the two approximation problems that have been studied [2, 7, 1]: Given a complete graph whose edges are labeled positive or negative, find a clustering that minimizes the number of disagreements, or maximizes the number of agreements. We will call these problems MinDisAgree and MaxAgree respectively. Bansal et al. [2] studied approximation algorithms both for minimization and maximization problems, giving a constant factor algorithm for MinDisAgree, and a *Polynomial Time Approximation Scheme (PTAS)* for MaxAgree. Charikar et al. [7] proved that MinDisAgree is APX-hard and gave a factor 4 approximation. Ailon et al. [1] presented a randomized factor 2.5 approximation for MinDisAgree, which is currently the best known factor. The problem has attracted significant attention, with further work on several variants [9, 6, 11, 13, 3, 12, 14].

In this paper, we study online algorithms for MinDisAgree and MaxAgree. We prove that MinDisAgree is essentially hopeless in the online setting: the natural greedy algorithm is $O(n)$-competitive, and this is optimal up to a constant factor, even with randomization (Theorem 3.4). The situation is better for MaxAgree: we prove that the greedy algorithm is a .5-competitive (Theorem 2.1), but that no algorithm can be better than 0.803 competitive (0.834 for randomized algorithms, see Theorem 2.2). What is the optimal competitive ratio? We prove that it is better than .5 by exhibiting an algorithm with competitive ratio $0.5 + \epsilon_0$ where ϵ_0 is a small absolute constant (Theorem 2.6). Thus Greedy is not always the best choice!

More formally, let $v_1, \dots, v_n$ denote the sequence of vertices of the input graph, where n is not known in advance. Between any two vertices, v_i and v_j for $i \neq j$, there is an edge labeled positive or negative. In MinDisAgree (resp. MaxAgree), the goal is to find a clustering $\mathcal{C}$, *i.e.* a partition of the nodes, that minimizes the number of disagreements $\mathrm{cost}(\mathcal{C})$: the number of negative edges within clusters plus the number of positive edges between clusters (resp. maximizes the number of agreements $\mathrm{profit}(\mathcal{C})$: the number of positive edges within clusters plus the number of negative edges between clusters). Although these problems are equivalent in terms of optimality, they differ from the point of view of approximation. Let OPT denote the optimum solution of MinDisAgree and of MaxAgree.

In the online setting, upon the arrival of a new vertex, the algorithm updates the current clustering: it may either create a new singleton cluster or add the new vertex to a pre-existing cluster, and may decide to merge some pre-existing clusters. It is not allowed to split pre-existing clusters.

A c-competitive algorithm for MinDisAgree outputs, on any input σ, a clustering $\mathcal{C}(\sigma)$ such that $\mathrm{cost}(\mathcal{C}(\sigma)) \leq c \cdot \mathrm{cost}(\mathrm{OPT}(\sigma))$. For MaxAgree, we must have $\mathrm{profit}(\mathcal{C}(\sigma)) \geq c \cdot \mathrm{profit}(\mathrm{OPT}(\sigma))$. (When the algorithm is randomized, this must hold in expectation).

2. Maximizing Agreements Online

2.1. Competitiveness of Greedy

For subsets of vertices S and T we define $\Gamma(S,T)$ as the set of edges between S and T. We write $\Gamma^+(S,T)$ (resp. $\Gamma^-(S,T)$) for the set of positive (resp. negative) edges of $\Gamma(S,T)$.

We define the *gain* of merging S with T as the change in the profit when clusters S and T are merged:

$$\text{gain}(S,T) = |\Gamma^{+}(S,T)| - |\Gamma^{-}(S,T)| = 2|\Gamma^{+}(S,T)| - |S||T|.$$

We present the following greedy algorithm for online correlation clustering.

Algorithm 1 Algorithm Greedy

```
Upon the arrival of vertex $v$ do
    Put $v$ in a new cluster consisting of $\{v\}$.
    while there are two clusters $C$, $C'$ such that $\text{gain}(C,C') > 0$ do
        Merge $C$ and $C'$
    end while
end for
```

Theorem 2.1. *Let OPT denote the offline optimum.*

- *For every instance, profit*(Greedy) ≥ 0.5 *profit*(OPT).
- *There are instances with profit*(Greedy) $\leq (0.5 + o(1))$*profit*(OPT).

2.2. Bounding the optimal competitive ratio

Theorem 2.2. *The competitive ratio of any randomized online algorithm for* MaxAgree *is at most* 0.834*. The competitive ratio of any deterministic online algorithm for* MaxAgree *is at most* 0.803*.*

The proof uses Yao's Min-Max Theorem [4] (maximization version).

Theorem 2.3 (Yao's Min-Max Theorem)**.** *Fix a distribution D over a set of inputs $(I_\sigma)_\sigma$. The competitive ratio of any randomized online algorithm is at most*

$$\max\{\frac{E_I[\mathit{profit}(\mathcal{A}(I))]}{E_I[\mathit{profit}(OPT(I))]} : \mathcal{A} \textit{ deterministic online algorithm}\},$$

where the expectations are over a random input I drawn from distribution D.

To prove Theorem 2.2, we first define two generic inputs that we will use to apply Theorem 2.3. The first input is a graph G_1 with $2m$ vertices and all positive edges between them The second input is a graph with $6m$ vertices defined as follows. The first $2m$ vertices have all positive edges between them, the next $2m$ vertices have all positive edges between them, and the last $2m$ vertices also have all positive edges between them. In each of these three sets G_1, G_2, G_3 of $2m$ vertices, half are labelled "left side" vertices and the other half are labelled "right side" vertices. All edges between left vertices are positive, but edges between a vertex u on the left side of G_i and a vertex v on the right side of G_j, $j \neq i$, are all negative.

The online algorithm cannot distinguish between the two inputs until time $2m+1$, so it must hedge against two very different possible optimal structures.

2.3. Beating Greedy

2.3.1. *Designing the algorithm.* Our algorithm is based on the observation that Algorithm Greedy always satisfies at least half of the edges. Thus, if profit(OPT) is less than $(1-\alpha/2)|E|$ for some constant α, then the profit of Greedy is better than half of optimal. We design an algorithm called Dense, parameterized by constants α and τ, such that if profit(OPT) is greater than $(1-\alpha/2)|E|$, then the approximation factor is at least $0.5+\eta$ for some positive constant η. We use both algorithms Greedy and Dense to define Algorithm 2.

Theorem 2.4. *Let $\alpha \in (0,1)$, $\tau > 1$ and $\eta \in (0,\frac{1}{2})$ be such that*

$$\eta \leq 1.5 - \tau^2 - ((2\sqrt{3}+9/2)\alpha^{1/4} + \frac{\alpha^{1/4}}{1-\alpha^{1/4}} + \alpha/2)2\frac{2\tau-1}{(\tau-1)}. \tag{2.1}$$

Then, for every instance such that $OPT \geq (1-\alpha/2)E$, Algorithm $\text{Dense}_{\alpha,\tau}$ has profit at least $(1/2+\eta)OPT$.

Using Theorem 2.4 we can bound the competitive ratio of Algorithm 2.

Corollary 2.5. *Let α, τ and η be as above, and let $p = \alpha/(2+2\eta(2-\alpha))$. Then Algorithm 2 has competitive ratio at least $\frac{1}{2} + \frac{\alpha\eta/2}{1+2\eta(1-\alpha/2)}$.*

Corollary 2.6. *For $\alpha = 10^{-12}$, $\tau = 1.0946$, $\eta = 0.0555$ and $p = 4,5 \cdot 10^{-13}$, Algorithm 2 is $\frac{1}{2} + 2 \cdot 10^{-14}$-competitive.*

Algorithm 2 A $\frac{1}{2}+\epsilon_0$-competitive algorithm

```
Given $p$, $\alpha$, $\tau$,
With probability $1-p$, run Greedy,
With probability $p$, run $\text{Dense}_{\alpha,\tau}$.
```

Algorithm 3 Algorithm $\text{Dense}_{\alpha,\tau}$

```
Let $\mathcal{C} = \widehat{\text{OPT}}_1$ and for every cluster $D \in \mathcal{C}$, let $\text{repr}_1(D) := D \in \widehat{\text{OPT}}_1$ .
Upon the arrival of a vertex $v$ at time $t$ do
    Put $v$ in a new cluster $\{v\}$.
    if $t = t_i$ for some $i$ then
        for every cluster $D$ in $\widehat{\text{OPT}_i}$ do
            Define a cluster $D''$ obtained by merging the restriction of $D$ to $\{t_{i-1},\dots,t_i\}$
            with every cluster $C \in \mathcal{C}$ in $\{1,\dots,t_{i-1}\}$ such that $\text{repr}_{i-1}(C)$ is defined and is
            half-contained in $D$.
            If $D''$ is not empty, set $\text{repr}_i(D'') := D \in \widehat{\text{OPT}_i}$.
        end for
    end if
end for
```

How do we define algorithm Dense? Using the PTAS of [2], one can compute offline a factor $(1-\alpha/2)$ approximative solution OPT' of any instance of MaxAgree in polynomial time. We will design algorithm Dense so that it guarantees an approximation factor of $0.5+\eta$ whenever $\text{profit}(\text{OPT}') \geq (1-\alpha)|E|$. Since $\text{profit}(\text{OPT}) \geq (1-\alpha/2)|E|$ implies that $\text{profit}(\text{OPT}') \geq (1-\alpha)|E|$, Theorem 2.4 will follow.

We say that OPT'_t is *large* if $\text{profit}(\text{OPT}'_t) \geq (1-\alpha)|E|$. We define a sequence $(t_i)_i$ of *update times* inductively as follows: By convention $t_0 = 0$. Time t_1 is the earliest time $t \geq 100$ such that OPT'_t is large. Assume t_i is already defined, and let j be such that $\tau^{j-1} \leq t_i < \tau^j$. If OPT'_{τ^j} is large, then $t_{i+1} = \tau^j$, else t_{i+1} is the earliest time $t \geq \tau^j$ such that OPT'_t is large. Let $t_1, t_2, \ldots, t_K$ be the resulting sequence. We will note, with an abuse of notation, OPT'_i instead of OPT'_{t_i} for $1 \leq i \leq K$.

We say that a cluster A is *half-contained* in B if $|A \cap B| > |A|/2$. Let $\epsilon = \alpha^{1/4}$. For each t_i, we inductively define a near optimal clustering of the nodes $[1, t_i]$. For the base case, let $\widehat{\text{OPT}}_1$ be the clustering obtained from OPT'_1 by keeping the $1/\epsilon^2$ largest clusters and splitting the other clusters into singletons. For the general case, to define $\widehat{\text{OPT}}_i$ given $\widehat{\text{OPT}}_{i-1}$, mark the clusters of OPT'_i as follows. For any D in OPT'_i, mark D if either one of the $1/\epsilon^2 - 1/\epsilon$ largest clusters of $\widehat{\text{OPT}}_{i-1}$ is half-contained in D, or D is one of the $1/\epsilon$ largest clusters OPT'_i. Then $\widehat{\text{OPT}}_i$ contains all the marked clusters of OPT'_i and the rest of the vertices in $[1, t_i]$ as singleton clusters. (Note that, by definition, any $\widehat{\text{OPT}}_i$ contains at most $1/\epsilon^2$ non-singleton clusters; this will be useful in the analysis.)

Note that Dense only depends on parameters α and τ indirectly via the definition of update times and of $\widehat{\text{OPT}}$.

2.3.2. *Analysis: Proof of Theorem 2.4.* The analysis is by induction on i, assuming that we start from clustering $\widehat{\text{OPT}}_i$ at time t_i, then apply the above algorithm from time t_i to the final time t. If $i = 1$ this is exactly our algorithm, and if $i = K$ then this is simply $\widehat{\text{OPT}}_K$; in general it is a mixture of the two constructions.

More formally, define a forest $\mathcal{F}$ (at time t) with one node for each $t_i \leq t$ and cluster of $\widehat{\text{OPT}}_i$. The node associated to a cluster A of $\widehat{\text{OPT}}_{i-1}$ is a child of the node associated to a cluster B of $\widehat{\text{OPT}}_i$ if and only if A is half-contained in B. With a slight abuse of notation, we define the following clustering $\mathcal{F}$ associated to the forest. There is one cluster T for each tree of the forest: for each node A of the tree, if i is such that $A \in \widehat{\text{OPT}}_i$, then cluster T contains $A \cap (t_{i-1}, t_i]$. This defines T.

One interpretation of Dense is that at all times t, there is an associated forest and clustering $\mathcal{F}$; and our algorithm Dense simply maintains it. See Figure 1 for an example.

Lemma 2.7. *Algorithm 3 is an online algorithm that outputs clustering $\mathcal{F}$ at time t.*

Let $\mathcal{F}_i$ be the forest obtained from $\mathcal{F}$ by erasing every node associated to clusters of $\widehat{\text{OPT}}_j$ for every $j < i$. With a slight abuse of notation, we define the following clustering $\mathcal{F}_i$ associated to that forest: there is one cluster C for each tree of the forest defined as follows. For each node A of the tree, let $k \geq i$ be such that $A \in \widehat{\text{OPT}}_k$: then C contains $A \cap (t_{k-1}, t_k]$ if $k > i$, and C contains A if $k = i$. This defines a sequence of clusterings such that $\mathcal{F}_1 = \mathcal{F}$ is the output of the algorithm, and $\mathcal{F}_K = \widehat{\text{OPT}}_K$.

Lemma 2.8 (Main lemma)**.** *For any $2 \leq i \leq K$,*

$$\textit{cost}(\mathcal{F}_{i-1}) - \textit{cost}(\mathcal{F}_i) \leq \left((4+2\sqrt{3})\epsilon + \frac{\epsilon}{1-\epsilon} \right) t_i t_K.$$

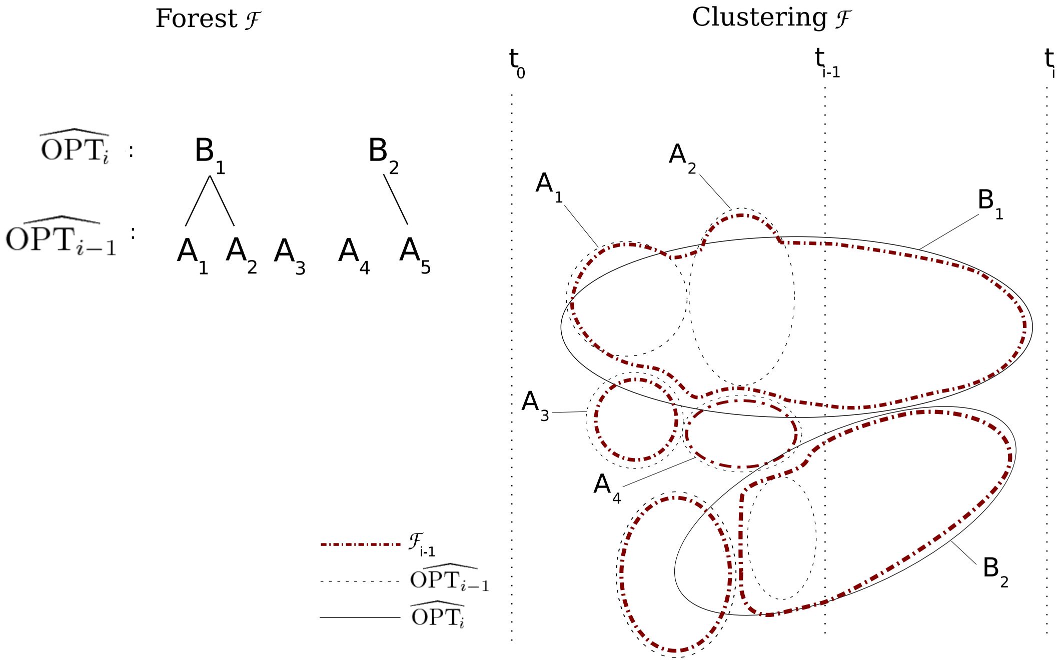

Forest $\mathcal{F}$
Clustering $\mathcal{F}$
$\widehat{\mathrm{OPT}}_i$:
$\widehat{\mathrm{OPT}}_{i-1}$:
B1
B2
A1
A2
A3
A4
A5
t0
ti-1
ti
$\mathcal{F}_{i-1}$
$\widehat{\mathrm{OPT}}_{i-1}$
$\widehat{\mathrm{OPT}}_i$

Figure 1: An example of a forest $\mathcal{F}$ given in left, and the corresponding clustering given in right. Here, we have $\widehat{\mathrm{OPT}}_i = \{B_1, B_2\}$ and $\widehat{\mathrm{OPT}}_{i-1} = \{A_1, \ldots, A_5\}$.

We defer the proof of Lemma 2.8 to next section. Assuming Lemma 2.8, we upperbound the cost of clustering $\mathcal{F}$.

Lemma 2.9 (Lemma 14, [2])**.** *For any $0 < c < 1$ and clustering $\mathcal{C}$, let $\mathcal{C}'$ be the clustering obtained from $\mathcal{C}$ by splitting all clusters of $\mathcal{C}$ of size less than cn, where n is the number of vertices. Then cost$(\mathcal{C}') \leq$ cost$(\mathcal{C}) + cn^2/2$.*

Lemma 2.10. *cost*$(\mathcal{F}) \leq ((2\sqrt{3} + 9/2)\epsilon + \frac{\epsilon}{1-\epsilon} + \epsilon^4/2)\frac{2\tau-1}{\tau-1}t_K^2$.

Proof. We write: $\mathrm{cost}(\mathcal{F}) = \mathrm{cost}(\widehat{\mathrm{OPT}}_K) + \sum_{i=2}^{K}(\mathrm{cost}(\mathcal{F}_{i-1}) - \mathrm{cost}(\mathcal{F}_i))$.By definition, $\widehat{\mathrm{OPT}}_K$ contains the $1/\epsilon$ largest clusters of OPT'_K. Then the remaining clusters of OPT'_K are of size at most ϵt_K. By Lemma 2.9, the cost of $\widehat{\mathrm{OPT}}_K$ is at most $\mathrm{cost}(\mathrm{OPT}'_K) + \epsilon t_K^2/2 \leq (\alpha + \epsilon)t_K^2/2$. Applying Lemma 2.8, and summing over $2 \leq i \leq K$, we get

$$\mathrm{cost}(\mathcal{F}) \leq (\alpha + \epsilon)t_K^2/2 + \left((4 + 2\sqrt{3})\epsilon + \frac{\epsilon}{1-\epsilon}\right)\sum_i t_i t_K.$$

By definition of the update times $(t_i)_i$, for any $j > 0$ there exists at most one t_i such that $\tau^j \leq t_i < \tau^{j+1}$. Let L be such that $\tau^L \leq t_K < \tau^{L+1}$. Then

$$\sum_{1 \leq i \leq K} t_i \leq \sum_{1 \leq i \leq K-1} t_i + t_K \leq \sum_{1 \leq j \leq L} \tau^j + t_K \leq \frac{\tau^{L+1}}{\tau - 1} + t_K \leq \frac{2\tau - 1}{\tau - 1} t_K.$$

Hence the desired bound on $\mathrm{cost}(\mathcal{F})$. ■

Proof of Theorem 2.4. Fix an input graph of size n, such that $\text{profit}(\text{OPT}) \geq (1-\alpha/2)\binom{n}{2}$. By Lemma 2.10, at time t_K, Algorithm 3 has clustering $\mathcal{F}$ with $\text{cost}(\mathcal{F}) \leq O(\epsilon)\frac{2\tau-1}{\tau-1}t_K^2$.

By definition of the update times, $n < \tau t_K$. To guarantee a competitive ratio of $0.5+\eta$, for some η, the cost must not exceed $(0.5-\eta)\binom{n}{2}$ at time n, when all vertices $t_K+1,\ldots,n$ are added as singleton clusters. The number of new edges added to the graph between times t_K and n is $\binom{n-t_K}{2} + t_K(n-t_K)$. We must have

$$\frac{2\tau-1}{\tau-1}O(\epsilon)t_K^2 + \binom{n-t_K}{2} + t_K(n-t_K) \leq (0.5-\eta)\binom{n}{2}, \tag{2.2}$$

for some $0<\eta<0.5$. Using the fact that $n-t_K \leq (\tau-1)t_K$ and $t_K \leq n-1$, to satisfy (2.2), it suffices to have

$$\frac{2\tau-1}{\tau-1}O(\epsilon)t_K^2 + t_K^2(\tau-1)^2/2 + (\tau-1)t_K^2 \leq (0.5-\eta)t_K^2/2,$$

which is equivalent to (2.1). Moreover we have the following natural constraints on constants η, ϵ and τ: $0<\eta<0.5$, $0<\epsilon<1$, and $\tau>1$. Then, for any set of values of constants η, ϵ, τ verifying those constraints, Algorithm Dense is $0.5+\eta$-competitive. ∎

2.3.3. *The core of the analysis: proof of Lemma 2.8.*

Lemma 2.11. *Let $\mathcal{S}^i$ be the set of vertices of the non-singleton clusters that are not among the $1/\epsilon^2 - 1/\epsilon$ largest clusters of $\widehat{OPT_{i-1}}$. Then $|\mathcal{S}^i| \leq \frac{\epsilon}{1-\epsilon}t_{i-1}$.*

Proof. Let C be a cluster of $\widehat{\text{OPT}_{i-1}}$, such that $C \subseteq \mathcal{S}^i$. Then $|C| \leq (1/\epsilon^2 - 1/\epsilon)^{-1}t_{i-1}$. Since there are at most $1/\epsilon$ such clusters, the number of vertices of these are at most $1/\epsilon(1/\epsilon^2-1/\epsilon)^{-1}t_{i-1}$. ∎

Notation 2.12. For any $i \neq j$, and a cluster B of OPT'_i, we denote by $\gamma_B^{i,j}$ the square root of the number of edges of $[1, t_{\min(i,j)}] \times [1, t_{\min(i,j)}]$, adjacent to at least one node of B, and which are classified differently in OPT'_i and in OPT'_j.

We refer to non singleton clusters as *large* clusters.

Lemma 2.13. *Let $\mathcal{T}^i$ be the set of vertices of those $1/\epsilon^2 - 1/\epsilon$ largest clusters of $\widehat{OPT_{i-1}}$ that are not half-contained in any cluster of OPT'_i. Then $|\mathcal{T}^i| \leq \sqrt{6}\sum_{\text{large } C\in\widehat{OPT_{i-1}}} \gamma_C^{i,i-1}$.*

Let B be a cluster of $\widehat{\text{OPT}_i}$. For any $j \leq i$, we define $\mathcal{C}_j(B)$ as the cluster associated with the tree of $\mathcal{F}_j$ that contains B. For any $B \in \widehat{\text{OPT}_i}$, we call $\mathcal{C}_{i-1}(B)$ the *extension* of $\mathcal{C}_i(B)$ to $\mathcal{F}_{i-1}$. By definition of $\mathcal{F}_i$, the following lemma is easy.

Lemma 2.14. *For any $B \in \widehat{OPT_i}$, the restriction of $\mathcal{C}_{i-1}(B)$ to $(t_{i-1}, t_K]$ is equal to the restriction of $\mathcal{C}_i(B)$ to $(t_{i-1}, t_K]$.*

Let $(A_j)_j$ denote the clusters of $\widehat{\text{OPT}_{i-1}}$ that are half-contained in B. We define $\delta^i(B)$ as the symmetric difference of the restriction of B to $[1, t_{i-1}]$ and $\cup_j A_j$:

$$\delta^i(B) = (B \cap [1, t_{i-1}])\Delta \cup_j A_j.$$

Lemma 2.15. *For any cluster C_i of $\mathcal{F}_i$, let C_i' denote the extension of C_i to $\mathcal{F}_{i-1}$. Then*

$$\bigcup_{C_i \in \mathcal{F}_i} C_i \setminus C_i' \subseteq \mathcal{S}^i \cup \mathcal{T}^i \cup \bigcup_{\textit{large } B \in \widehat{OPT_i}} \delta^i(B)$$

Proof. By Lemma 2.14, the partition of the vertices $(t_{i-1}, t_K]$ is the same for C_i as for C_i'. So C_i and C_i' only differ in the vertices of $[1, t_{i-1})$:

$$\bigcup_{C_i \in \mathcal{F}_i} C_i \setminus C_i' \subseteq \bigcup_{B \in \widehat{\mathrm{OPT}_i}} \delta^i(B).$$

We will show that for a singleton cluster B of $\widehat{\mathrm{OPT}_i}$, $\delta^i(B)$ is included in $\mathcal{S}^i \cup \mathcal{T}^i \bigcup_{\text{large } B \in \widehat{\mathrm{OPT}_i}} \delta^i(B)$, which yields the lemma.

Let $B = \{v\}$ be a singleton cluster of $\widehat{\mathrm{OPT}_i}$ such that $\delta^i(B) \neq \{\}$. A non-singleton cluster cannot be half-contained in a singleton cluster so we conclude no clusters are half-contained in B and hence $\delta^i(B) = \{v\}$. By definition of $\delta^i(B)$, $v \in [1, t_{i-1}]$. So there exists a cluster A of $\widehat{\mathrm{OPT}_{i-1}}$ that contains v. Clearly A is not a singleton since otherwise $\delta^i(B)$ would be $\{\}$. There are two cases.

First, if A is half-contained in a cluster $B' \neq B$ of $\widehat{\mathrm{OPT}_i}$ then cluster B' is necessarily large since it contains more than one vertex of A. Then we have $v \in \delta^i(B')$.

Second, if A is not half-contained in any cluster of $\widehat{\mathrm{OPT}_i}$ then $A \subseteq \mathcal{S}^i \cup \mathcal{T}^i$. In fact, if A is half-contained in a cluster of OPT_i' which is split into singletons in $\widehat{\mathrm{OPT}_i}$, then A is not one of the $1/\epsilon^2 - 1/\epsilon$ largest clusters of $\widehat{\mathrm{OPT}_{i-1}}$, and $A \subseteq \mathcal{S}^i$. If A is not half-contained in any cluster of OPT_i', then $A \subseteq \mathcal{T}^i$ if A is one of the $1/\epsilon^2 - 1/\epsilon$ largest clusters of $\widehat{\mathrm{OPT}_{i-1}}$ and $A \subseteq \mathcal{S}^i$ otherwise. ∎

Lemma 2.16. *For any large cluster B of $\widehat{OPT_i}$, $|\delta^i(B)| \leq 2\sqrt{2}\gamma_B^{i,i-1}$.*

Proof. Let B' denote the restriction of B to $[1, t_{i-1}]$. We first show that

$$1/2(|\cup_j A_j \setminus B'|)^2 \leq (\gamma_B^{i,i-1})^2.$$

Observe that $(\gamma_B^{i,i-1})^2$ includes all edges uv such that one of the following two cases occurs.

First, if $u \in A_j \setminus B$ and $v \in A_j \cap B$: such edges are internal in the clustering OPT_{i-1}' but external in the clustering OPT_i'. The number of edges of this type is $\sum_j |A_j \setminus B| \cdot |A_j \cap B|$. Since A_j is half-contained in B, this is at least $\sum_j |A_j \setminus B|^2$.

Second, if $u \in A_j \cap B$ and $v \in A_k \cap B$ with $j \neq k$: such edges are external in the clustering OPT_{i-1}' but internal in the clustering OPT_i'. The number of edges of this type is $\sum_{j<k} |A_j \cap B| \cdot |A_k \cap B| \geq \sum_{j<k} |A_j \setminus B| \cdot |A_k \setminus B|$.

Summing, it is easy to infer that $(\gamma_B^{i,i-1})^2 \geq (1/2) \left(\sum_j |A_j \setminus B|\right)^2 = (1/2)|\cup_j A_j \setminus B'|^2$.

Let $(A_j')_j$ denote the clusters of $\widehat{\mathrm{OPT}_{i-1}}$ that are not half-contained in B, but have nonempty intersections with B. We now show that

$$1/2(|B' \setminus \cup_j A_j'|)^2 \leq (\gamma_B^{i,i-1})^2.$$

We have $B' \setminus \cup_j A_j = \cup_j (A_j' \cap B)$. Observe that any A_j' is a large cluster of $\widehat{\mathrm{OPT}_{i-1}}$, thus a cluster of OPT_{i-1}'. Then $(\gamma_B^{i,i-1})^2$ includes all edges uv such that one of the following two cases occurs

First, if $u \in A'_j \setminus B$ and $v \in A'_j \cap B$: such edges are internal in the clustering OPT'_{i-1} but external in the clustering OPT'_i. The number of edges of this type is $\sum_j |A'_j \setminus B| \cdot |A'_j \cap B|$. Since A'_j is not half-contained in B, this is at least $\sum_j |A'_j \cap B|^2$.

Second, if $u \in A'_j \cap B$ and $v \in A'_k \cap B$ with $j \neq k$: such edges are external in the clustering OPT'_{i-1} but internal in the clustering OPT'_i. The number of edges of this type is $\sum_{j<k} |A'_j \cap B| \cdot |A'_k \cap B|$.

Summing, we get

$$(\gamma_B^{i,i-1})^2 \geq (1/2) \left(\sum_j |A'_j \cap B| \right)^2 = (1/2)|B' \setminus \cup_j A'_j|^2.$$

■

Lemma 2.17. *For any $i \geq 1$, $\widehat{OPT_i}$ has at most $1/\epsilon^2$ non singleton clusters, all of which are clusters of OPT'_i*

Proof. By definition, $\widehat{\mathrm{OPT}_1}$ has at most $1/\epsilon^2$ non singleton clusters. For any $i > 1$, a cluster of $\widehat{\mathrm{OPT}_{i-1}}$ can only be half-contained in one cluster of OPT'_i. Therefore given $\widehat{\mathrm{OPT}_{i-1}}$, at most $1/\epsilon^2$ clusters of OPT'_i are marked. Thus $\widehat{\mathrm{OPT}_i}$ has at most $1/\epsilon^2$ clusters. ■

We can now prove Lemma 2.8.

Proof of Lemma 2.8. By Lemma 2.14, clusterings $\mathcal{F}_i$ and $\mathcal{F}_{i-1}$ only differ in their partition of $[1, t_{i-1}]$. Then the set of the vertices that are classified differently in $\mathcal{F}_i$ and $\mathcal{F}_{i-1}$ is $\cup_i C_i \setminus C_{i-1}$. Each of these vertices creates at most t_K disagreements:

$$\mathrm{cost}(\mathcal{F}_{i-1}) - \mathrm{cost}(\mathcal{F}_i) \leq \sum_{C_i \in \mathcal{F}_i} |C_i \setminus C_{i-1}| t_K \tag{2.3}$$

By Lemmas 2.15 and 2.16,

$$\sum_{C_i \in \mathcal{F}_i} |C_i \setminus C_{i-1}| t_K \leq \left(2\sqrt{2} \left(\sum_{\text{large } B \in \widehat{\mathrm{OPT}_i}} \gamma_B^{i,i-1} \right) + |\mathcal{S}^i| + |\mathcal{T}^i| \right) t_K. \tag{2.4}$$

By Lemmas 2.11 and 2.13,

$$|\mathcal{S}^i| \leq \frac{\epsilon}{1-\epsilon} t_{i-1} \text{ and } |\mathcal{T}^i| \leq \sqrt{6} \sum_{\text{large } B \in \widehat{\mathrm{OPT}_{i-1}}} \gamma_B^{i-1,i} \tag{2.5}$$

The term $\sum_{\text{large } B \in \widehat{\mathrm{OPT}_{i-1}}} \gamma_B^{i-1,i}$ can be seen as the ℓ_1 norm of the vector $(\gamma_B^{i-1,i})_{\text{large } B}$. Since $\widehat{\mathrm{OPT}_{i-1}}$ has at most $1/\epsilon^2$ large clusters by Lemma 2.17, we can use Hölder's inequality:

$$\sum_{\text{large } B \in \widehat{\mathrm{OPT}_{i-1}}} \gamma_B^{i-1,i} = \|(\gamma_B^{i-1,i})_{\text{large } B}\|_1 \leq 1/\epsilon \|(\gamma_B^{i-1,i})_{\text{large } B}\|_2.$$

By definition we have $\|(\gamma_B^{i-1,i})_{\text{large B}}\|_2 \leq \sqrt{2(\mathrm{cost}(\mathrm{OPT}'_{i-1}) + \mathrm{cost}(\mathrm{OPT}'_i))}$. Thus

$$\sum_{\text{large } B \in \widehat{\mathrm{OPT}_{i-1}}} \gamma_B^{i-1,i} \leq 1/\epsilon \sqrt{2(\alpha t_{i-1}^2/2 + \alpha t_i^2/2)} \leq \frac{\sqrt{2\alpha}}{\epsilon} t_i. \tag{2.6}$$

Similarly, we have

$$\sum_{\text{large } B\in\widehat{\mathrm{OPT}_i}} \gamma_B^{i,i-1} \le \frac{\sqrt{2\alpha}}{\epsilon} t_i. \tag{2.7}$$

Combining equations (2.3) through (2.7) and $\alpha = \epsilon^4$ yields

$$\mathrm{cost}(\mathcal{F}_{i-1}) - \mathrm{cost}(\mathcal{F}_i) \le \left((4+2\sqrt{3})\epsilon + \frac{\epsilon}{1-\epsilon} \right) t_i t_K$$

■

3. Minimizing Disagreements Online

Theorem 3.1. *Algorithm* Greedy *is* $(2n+1)$*-competitive for* MinDisAgree.

To prove Theorem 3.1, we need to compare the cost of the optimal clustering to the cost of the clustering constructed by the algorithm. The following lemma reduces this to, roughly, analyzing the number of vertices classified differently.

Lemma 3.2. *Let $\mathcal{W}$ and $\mathcal{W}'$ be two clusterings such that there is an injection $W_i' \in \mathcal{W}' \to W_i \in \mathcal{W}$. Then cost($\mathcal{W}'$) − cost($\mathcal{W}$) $\le n\sum_i |W_i' \setminus W_i|$.*

For subsets of vertices $S_1, \ldots, S_m$, we will write, with a slight abuse of notation, $\Gamma^+(S_1, \ldots, S_m)$ for the set of edges in $\Gamma^+(S_i, S_j)$ for any $i \neq j$: $\Gamma^+(S_1, \ldots, S_m) = \cup_{i\neq j}\Gamma^+(S_i, S_j)$.

Lemma 3.3. *Let C be a cluster created by* Greedy*, and $\mathcal{W} = \{W_1, \ldots, W_K\}$ denote the clusters of OPT. Then $|C| \le \max_i |C \cap W_i| + 2|\Gamma^+(C\cap W_1, \ldots, C\cap W_K)|$.. We call $i_0 = \arg\max_i |C\cap W_i|$ the* leader *of C.*

Proof of Theorem 3.1. Let $\mathcal{C}$ denote the clustering given by Greedy. For every cluster W_i of OPT, merge all the clusters of $\mathcal{C}$ that have i as their leaders. Let $\mathcal{C}' = (W_i')$ be this new clustering. By definition of the greedy algorithm, this operation can only increase the cost since every pair of clusters have a negative-majority cut at the end of the algorithm:cost($\mathcal{C}$) $\le$ cost($\mathcal{C}'$). We apply Lemma 3.2 to $\mathcal{W}$ =OPT and $\mathcal{W}' = \mathcal{C}'$, and obtain: cost($\mathcal{C}'$) $\le$ cost(OPT) $+ n\sum_i |W_i' \setminus W_i|$. By definition of $\mathcal{C}'$ we have $|W_i' \setminus W_i| = \sum_{C\in\mathcal{C}:\mathrm{leader}(C)=i}\sum_{j\neq i}|C\cap W_j|$, hence

$$\sum_i |W_i' \setminus W_i| = \sum_{C\in\mathcal{C}} \sum_{j\neq \mathrm{leader}(C)} |C\cap W_j|.$$

By Lemma 3.3, $\sum_{j\neq\mathrm{leader}(C)}|C\cap W_j| \le 2|\Gamma^+(C\cap W_1, \ldots, C\cap W_K)|$. Finally, to bound OPT from below, we observe that, for any two clusterings $\mathcal{C}$ and $\mathcal{W}$, it holds that the sum over $C \in \mathcal{C}$ of $|\Gamma^+(C\cap W_1, \ldots, C\cap W_K)|$ is less than cost($\mathcal{W}$). Combining these inequalities yields the theorem. ■

Theorem 3.4. *Let ALG be a randomized algorithm for* MinDisAgree. *Then there exists an instance on which ALG has cost at least $n - 1 - cost(OPT)$ where OPT is the offline optimum. If OPT is constant then $cost(ALG) = \Omega(n) cost(OPT)$.*

Proof. Consider two cliques A and B, each of size m, where all the internal edges of A and B are positive. Choose a vertex a in A, and a set of vertices $b_1, \ldots, b_k$ in B. Define the edge labels of ab_i as positive, for all $1 \leq i \leq k$ and the rest of the edges between A and B as negative. Define an input sequence starting with $a, b_1, \ldots, b_k$, followed by the rest of the vertices in any order.

■

References

[1] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 684–693, New York, NY, USA, 2005. ACM Press.

[2] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, 2004.

[3] Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3-4):281–297, 1999.

[4] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[5] Moses Charikar, Chandra Chekuri, Tomas Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004.

[6] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. In *focs*, volume 00, page 524, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[7] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005.

[8] William W. Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 475–480, New York, NY, USA, 2002. ACM.

[9] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2):172–187, 2006.

[10] Jenny Rose Finkel and Christopher D. Manning. Enforcing transitivity in coreference resolution. In *Proceedings of ACL-08: HLT, Short Papers*, pages 45–48, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[11] Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. *Theory of Computing*, 2(1):249–266, 2006.

[12] Thorsten Joachims and John Hopcroft. Error bounds for correlation clustering. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 385–392, New York, NY, USA, 2005. ACM.

[13] Marek Karpinski and Warren Schudy. Linear time approximation schemes for the Gale-Berlekamp game and related minimization problems. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 313–322, 2009.

[14] Claire Mathieu and Warren Schudy. Correlation clustering with noisy input. In *To appear in Procs. 21st SODA, preprint: http://www.cs.brown.edu/∼ws/papers/cluster.pdf*, 2010.

[15] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Appl. Math.*, 144(1-2):173–182, 2004.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 585-596
www.stacs-conf.org

THE RECOGNITION OF TOLERANCE AND BOUNDED TOLERANCE GRAPHS

GEORGE B. MERTZIOS[1] AND IGNASI SAU[2] AND SHMUEL ZAKS[3]

[1] Department of Computer Science, RWTH Aachen University, Aachen, Germany
E-mail address: mertzios@cs.rwth-aachen.de

[2] Department of Computer Science, Technion, Haifa, Israel
E-mail address: ignasi.sau@gmail.com

[3] Department of Computer Science, Technion, Haifa, Israel
E-mail address: zaks@cs.technion.ac.il

Abstract. Tolerance graphs model interval relations in such a way that intervals can tolerate a certain degree of overlap without being in conflict. This subclass of perfect graphs has been extensively studied, due to both its interesting structure and its numerous applications. Several efficient algorithms for optimization problems that are NP-hard on general graphs have been designed for tolerance graphs. In spite of this, the recognition of tolerance graphs – namely, the problem of deciding whether a given graph is a tolerance graph – as well as the recognition of their main subclass of bounded tolerance graphs, have been the most fundamental open problems on this class of graphs (cf. the book on tolerance graphs [14]) since their introduction in 1982 [11]. In this article we prove that both recognition problems are NP-complete, even in the case where the input graph is a trapezoid graph. The presented results are surprising because, on the one hand, most subclasses of perfect graphs admit polynomial recognition algorithms and, on the other hand, bounded tolerance graphs were believed to be efficiently recognizable as they are a natural special case of trapezoid graphs (which can be recognized in polynomial time) and share a very similar structure with them. For our reduction we extend the notion of an *acyclic orientation* of permutation and trapezoid graphs. Our main tool is a new algorithm that uses *vertex splitting* to transform a given trapezoid graph into a permutation graph, while preserving this new acyclic orientation property. This method of vertex splitting is of independent interest; very recently, it has been proved a powerful tool also in the design of efficient recognition algorithms for other classes of graphs [21].

1. Introduction

1.1. Tolerance graphs and related graph classes

A simple undirected graph $G = (V, E)$ on n vertices is a *tolerance* graph if there exists a collection $I = \{I_i \mid i = 1, 2, \ldots, n\}$ of closed intervals on the real line and a set

1998 ACM Subject Classification: F.2.2 Computations on discrete structures, G.2.2 Graph algorithms.

Key words and phrases: Tolerance graphs, bounded tolerance graphs, recognition, vertex splitting, NP-complete, trapezoid graphs, permutation graphs.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2487

$t = \{t_i \mid i = 1, 2, \ldots, n\}$ of positive numbers, such that for any two vertices $v_i, v_j \in V$, $v_i v_j \in E$ if and only if $|I_i \cap I_j| \geq \min\{t_i, t_j\}$. The pair $\langle I, t\rangle$ is called a *tolerance representation* of G. If G has a tolerance representation $\langle I, t\rangle$, such that $t_i \leq |I_i|$ for every $i = 1, 2, \ldots, n$, then G is called a *bounded tolerance* graph and $\langle I, t\rangle$ a *bounded tolerance representation* of G.

Tolerance graphs were introduced in [11], in order to generalize some of the well known applications of interval graphs. The main motivation was in the context of resource allocation and scheduling problems, in which resources, such as rooms and vehicles, can tolerate sharing among users [14]. If we replace in the definition of tolerance graphs the operator *min* by the operator *max*, we obtain the class of *max-tolerance* graphs. Both tolerance and max-tolerance graphs find in a natural way applications in biology and bioinformatics, as in the comparison of DNA sequences from different organisms or individuals [17], by making use of a software tool like BLAST [1]. Tolerance graphs find numerous other applications in constrained-based temporal reasoning, data transmission through networks to efficiently scheduling aircraft and crews, as well as contributing to genetic analysis and studies of the brain [13,14]. This class of graphs has attracted many research efforts [2,4,8,12–15,18,22,24], as it generalizes in a natural way both interval graphs (when all tolerances are equal) and permutation graphs (when $t_i = |I_i|$ for every $i = 1, 2, \ldots, n$) [11]. For a detailed survey on tolerance graphs we refer to [14].

A *comparability* graph is a graph which can be transitively oriented. A *co-comparability* graph is a graph whose complement is a comparability graph. A *trapezoid* (resp. *parallelogram* and *permutation*) graph is the intersection graph of trapezoids (resp. parallelograms and line segments) between two parallel lines L_1 and L_2 [10]. Such a representation with trapezoids (resp. parallelograms and line segments) is called a *trapezoid* (resp. *parallelogram* and *permutation*) *representation* of this graph. A graph is bounded tolerance if and only if it is a parallelogram graph [2, 19]. Permutation graphs are a strict subset of parallelogram graphs [3]. Furthermore, parallelogram graphs are a strict subset of trapezoid graphs [25], and both are subsets of co-comparability graphs [10, 14]. On the contrary, tolerance graphs are not even co-comparability graphs [10, 14]. Recently, we have presented in [22] a natural intersection model for general tolerance graphs, given by parallelepipeds in the three-dimensional space. This representation generalizes the parallelogram representation of bounded tolerance graphs, and has been used to improve the time complexity of minimum coloring, maximum clique, and weighted independent set algorithms on tolerance graphs [22].

Although tolerance and bounded tolerance graphs have been studied extensively, the recognition problems for both these classes have been the most fundamental open problems since their introduction in 1982 [5, 10, 14]. Therefore, all existing algorithms assume that, along with the input tolerance graph, a tolerance representation of it is given. The only result about the complexity of recognizing tolerance and bounded tolerance graphs is that they have a (non-trivial) polynomial sized tolerance representation, hence the problems of recognizing tolerance and bounded tolerance graphs are in the class NP [15]. Recently, a linear time recognition algorithm for the subclass of *bipartite tolerance* graphs has been presented in [5]. Furthermore, the class of trapezoid graphs (which strictly contains parallelogram, i.e. bounded tolerance, graphs [25]) can be also recognized in polynomial time [20,21,26]. On the other hand, the recognition of max-tolerance graphs is known to be NP-hard [17]. Unfortunately, the structure of max-tolerance graphs differs significantly from that of tolerance graphs (max-tolerance graphs are not even perfect, as they can contain induced C_5's [17]), so the technique used in [17] does not carry over to tolerance graphs.

Since very few subclasses of perfect graphs are known to be NP-hard to recognize, it was believed that the recognition of tolerance graphs was in P. Furthermore, as bounded tolerance graphs are equivalent to parallelogram graphs [2, 19], which constitute a natural subclass of trapezoid graphs and have a very similar structure, it was plausible that their recognition was also in P.

1.2. Our contribution

In this article, we establish the complexity of recognizing tolerance and bounded tolerance graphs. Namely, we prove that both problems are surprisingly NP-complete, by providing a reduction from the monotone-Not-All-Equal-3-SAT (monotone-NAE-3-SAT) problem. Consider a boolean formula ϕ in conjunctive normal form with three literals in every clause (3-CNF), which is monotone, i.e. no variable is negated. The formula ϕ is called NAE-satisfiable if there exists a truth assignment of the variables of ϕ, such that every clause has at least one true variable and one false variable. Given a monotone 3-CNF formula ϕ, we construct a trapezoid graph H_ϕ, which is parallelogram, i.e. bounded tolerance, if and only if ϕ is NAE-satisfiable. Moreover, we prove that the constructed graph H_ϕ is tolerance if and only if it is bounded tolerance. Thus, since the recognition of tolerance and of bounded tolerance graphs are in the class NP [15], it follows that these problems are both NP-complete. Actually, our results imply that the recognition problems remain NP-complete even if the given graph is trapezoid, since the constructed graph H_ϕ is trapezoid.

For our reduction we extend the notion of an *acyclic orientation* of permutation and trapezoid graphs. Our main tool is a new algorithm that transforms a given trapezoid graph into a permutation graph by *splitting* some specific vertices, while preserving this new acyclic orientation property. One of the main advantages of this algorithm is its robustness, in the sense that the constructed permutation graph does not depend on any particular trapezoid representation of the input graph G. Moreover, besides its use in the present paper, this approach based on splitting vertices has been recently proved a powerful tool also in the design of efficient recognition algorithms for other classes of graphs [21].

Organization of the paper. We first present in Section 2 several properties of permutation and trapezoid graphs, as well as the algorithm Split-U, which constructs a permutation graph from a trapezoid graph. In Section 3 we present the reduction of the monotone-NAE-3-SAT problem to the recognition of bounded tolerance graphs. In Section 4 we prove that this reduction can be extended to the recognition of general tolerance graphs. Finally, we discuss the presented results and further research directions in Section 5. Some proofs have been omitted due to space limitations; a full version can be found in [23].

2. Trapezoid graphs and representations

In this section we first introduce (in Section 2.1) the notion of an *acyclic representation* of permutation and of trapezoid graphs. This is followed (in Section 2.2) by some structural properties of trapezoid graphs, which will be used in the sequel for the splitting algorithm Split-U. Given a trapezoid graph G and a vertex subset U of G with certain properties, this algorithm constructs a permutation graph $G^{\#}(U)$ with $2|U|$ vertices, which is independent on any particular trapezoid representation of the input graph G.

Notation. We consider in this article simple undirected and directed graphs with no loops or multiple edges. In an undirected graph G, the edge between vertices u and v is denoted by uv, and in this case u and v are said to be *adjacent* in G. If the graph G is directed, we denote by uv the arc from u to v. Given a graph $G = (V, E)$ and a subset $S \subseteq V$, $G[S]$ denotes the induced subgraph of G on the vertices in S, and we use $E[S]$ to denote $E(G[S])$. Whenever we deal with a trapezoid (resp. permutation and bounded tolerance, i.e. parallelogram) graph, we will consider w.l.o.g. a trapezoid (resp. permutation and parallelogram) representation, in which all endpoints of the trapezoids (resp. line segments and parallelograms) are distinct [9, 14, 16]. Given a permutation graph P along with a permutation representation R, we may not distinguish in the following between a vertex of P and the corresponding line segment in R, whenever it is clear from the context. Furthermore, with a slight abuse of notation, we will refer to the line segments of a permutation representation just as *lines.*

2.1. Acyclic permutation and trapezoid representations

Let $P = (V, E)$ be a permutation graph and R be a permutation representation of P. For a vertex $u \in V$, denote by $\theta_R(u)$ the angle of the line of u with L_2 in R. The class of permutation graphs is the intersection of comparability and co-comparability graphs [10]. Thus, given a permutation representation R of P, we can define two partial orders $(V, <_R)$ and $(V, \ll_R)$ on the vertices of P [10]. Namely, for two vertices u and v of G, $u <_R v$ if and only if $uv \in E$ and $\theta_R(u) < \theta_R(v)$, while $u \ll_R v$ if and only if $uv \notin E$ and u lies to the left of v in R. The partial order $(V, <_R)$ implies a transitive orientation Φ_R of P, such that $uv \in \Phi_R$ whenever $u <_R v$.

Let $G = (V, E)$ be a trapezoid graph, and R be a trapezoid representation of G, where for any vertex $u \in V$, the trapezoid corresponding to u in R is denoted by T_u. Since trapezoid graphs are also co-comparability graphs [10], we can similarly define the partial order $(V, \ll_R)$ on the vertices of G, such that $u \ll_R v$ if and only if $uv \notin E$ and T_u lies completely to the left of T_v in R. In this case, we may denote also $T_u \ll_R T_v$.

In a given trapezoid representation R of a trapezoid graph G, we denote by $l(T_u)$ and $r(T_u)$ the left and the right line of T_u in R, respectively. Similarly to the case of permutation graphs, we use the relation $\ll_R$ for the lines $l(T_u)$ and $r(T_u)$, e.g. $l(T_u) \ll_R r(T_v)$ means that the line $l(T_u)$ lies to the left of the line $r(T_v)$ in R. Moreover, if the trapezoids of all vertices of a subset $S \subseteq V$ lie completely to the left (resp. right) of the trapezoid T_u in R, we write $R(S) \ll_R T_u$ (resp. $T_u \ll_R R(S)$). Note that there are several trapezoid representations of a particular trapezoid graph G. Given one such representation R, we can obtain another one R' by *vertical axis flipping* of R, i.e. R' is the mirror image of R along an imaginary line perpendicular to L_1 and L_2. Moreover, we can obtain another representation R'' of G by *horizontal axis flipping* of R, i.e. R'' is the mirror image of R along an imaginary line parallel to L_1 and L_2. We will extensively use these two operations throughout the article.

Definition 2.1. Let P be a permutation graph with $2n$ vertices $\{u_1^1, u_1^2, u_2^1, u_2^2, \ldots, u_n^1, u_n^2\}$. Let R be a permutation representation and Φ_R be the corresponding transitive orientation of P. The simple directed graph F_R is obtained by merging u_i^1 and u_i^2 into a single vertex u_i, for every $i = 1, 2, \ldots, n$, where the arc directions of F_R are implied by the corresponding directions in Φ_R. Then,

(1) R is an *acyclic permutation representation with respect to* $\{u_i^1, u_i^2\}_{i=1}^n$[*], if F_R has no directed cycle,

(2) P is an *acyclic permutation graph with respect to* $\{u_i^1, u_i^2\}_{i=1}^n$, if P has an acyclic representation R with respect to $\{u_i^1, u_i^2\}_{i=1}^n$.

Definition 2.2. Let G be a trapezoid graph with n vertices and R be a trapezoid representation of G. Let P be the permutation graph with $2n$ vertices corresponding to the left and right lines of the trapezoids in R, R_P be the permutation representation of P induced by R, and $\{u_i^1, u_i^2\}$ be the vertices of P that correspond to the same vertex u_i of G, $i = 1, 2, \ldots, n$. Then,

(1) R is an *acyclic trapezoid representation*, if R_P is an acyclic permutation representation with respect to $\{u_i^1, u_i^2\}_{i=1}^n$,

(2) G is an *acyclic trapezoid graph*, if it has an acyclic representation R.

The following lemma follows easily from Definitions 2.1 and 2.2.

Lemma 2.3. *Any parallelogram graph is an acyclic trapezoid graph.*

2.2. Structural properties of trapezoid graphs

In the following, we state some definitions concerning an arbitrary simple undirected graph $G = (V, E)$, which are useful for our analysis. Although these definitions apply to any graph, we will use them only for trapezoid graphs. Similar definitions, for the restricted case where the graph G is connected, were studied in [6]. For $u \in V$ and $U \subseteq V$, $N(u) = \{v \in V \mid uv \in E\}$ is the set of adjacent vertices of u in G, $N[u] = N(u) \cup \{u\}$, and $N(U) = \bigcup_{u \in U} N(u) \setminus U$. If $N(U) \subseteq N(W)$ for two vertex subsets U and W, then U is said to be *neighborhood dominated* by W. Clearly, the relationship of neighborhood domination is transitive.

Let $C_1, C_2, \ldots, C_\omega$, $\omega \geq 1$, be the connected components of $G \setminus N[u]$ and $V_i = V(C_i)$, $i = 1, 2, \ldots, \omega$. For simplicity of the presentation, we will identify in the sequel the component C_i and its vertex set V_i, $i = 1, 2, \ldots, \omega$. For $i = 1, 2, \ldots, \omega$, the *neighborhood domination closure* of V_i with respect to u is the set $D_u(V_i) = \{V_p \mid N(V_p) \subseteq N(V_i),\ p = 1, 2, \ldots, \omega\}$ of connected components of $G \setminus N[u]$. A component V_i is called a *master component* of u if $|D_u(V_i)| \geq |D_u(V_j)|$ for all $j = 1, 2, \ldots, \omega$. The *closure complement* of the neighborhood domination closure $D_u(V_i)$ is the set $D_u^*(V_i) = \{V_1, V_2, \ldots, V_\omega\} \setminus D_u(V_i)$. Finally, for a subset $S \subseteq \{V_1, V_2, \ldots, V_\omega\}$, a component $V_j \in S$ is called *maximal* if there is no component $V_k \in S$ such that $N(V_j) \subsetneqq N(V_k)$.

For example, consider the trapezoid graph G with vertex set $\{u, u_1, u_2, u_3, v_1, v_2, v_3, v_4\}$, which is given by the trapezoid representation R of Figure 1. The connected components of $G \setminus N[u] = \{v_1, v_2, v_3, v_4\}$ are $V_1 = \{v_1\}$, $V_2 = \{v_2\}$, $V_3 = \{v_3\}$, and $V_4 = \{v_4\}$. Then, $N(V_1) = \{u_1\}$, $N(V_2) = \{u_1, u_3\}$, $N(V_3) = \{u_2, u_3\}$, and $N(V_4) = \{u_3\}$. Hence, $D_u(V_1) = \{V_1\}$, $D_u(V_2) = \{V_1, V_2, V_4\}$, $D_u(V_3) = \{V_3, V_4\}$, and $D_u(V_4) = \{V_4\}$; thus, V_2 is the only master component of u. Furthermore, $D_u^*(V_1) = \{V_2, V_3, V_4\}$, $D_u^*(V_2) = \{V_3\}$, $D_u^*(V_3) = \{V_1, V_2\}$, and $D_u^*(V_4) = \{V_1, V_2, V_3\}$.

[*] To simplify the presentation, we use throughout the paper $\{u_i^1, u_i^2\}_{i=1}^n$ to denote the set of n unordered pairs $\{u_1^1, u_1^2\}, \{u_2^1, u_2^2\}, \ldots, \{u_n^1, u_n^2\}$.

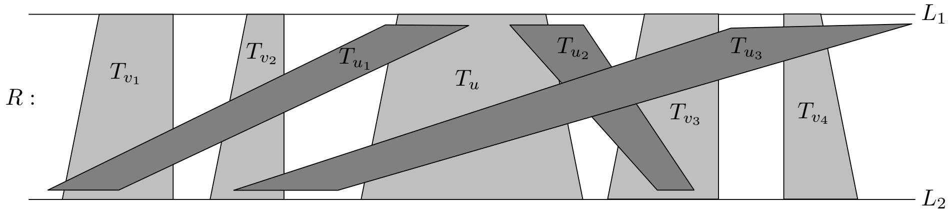


Figure 1: A trapezoid representation R of a trapezoid graph G.

Lemma 2.4. *Let G be a simple graph, u be a vertex of G, and let $V_1, V_2, \ldots, V_\omega$, $\omega \geq 1$, be the connected components of $G \setminus N[u]$. If V_i is a master component of u, such that $D^*_u(V_i) \neq \emptyset$, then $D^*_u(V_j) \neq \emptyset$ for every component V_j of $G \setminus N[u]$.*

In the following we investigate several properties of trapezoid graphs, in order to derive the vertex-splitting algorithm Split-U in Section 2.3.

Remark 2.5. Similar properties of trapezoid graphs have been studied in [6], leading to another vertex-splitting algorithm, called Split-All. However, the algorithm proposed in [6] is incorrect, since it is based on an incorrect property[†], as was also verified by [7]. In the sequel of this section, we present new definitions and properties. In the cases where a similarity arises with those of [6], we refer to it specifically.

Lemma 2.6. *Let R be a trapezoid representation of a trapezoid graph G, and V_i be a master component of a vertex u of G, such that $R(V_i) \ll_R T_u$. Then, $T_u \ll_R R(V_j)$ for every component $V_j \in D^*_u(V_i)$.*

Definition 2.7. Let G be a trapezoid graph, u be a vertex of G, and V_i be an arbitrarily chosen master component of u. Then, $\delta_u = V_i$ and

(1) if $D^*_u(V_i) = \emptyset$, then $\delta^*_u = \emptyset$.
(2) if $D^*_u(V_i) \neq \emptyset$, then $\delta^*_u = V_j$, for an arbitrarily chosen maximal component $V_j \in D^*_u(V_i)$.

Actually, as we will show in Lemma 2.10, the arbitrary choice of the components V_i and V_j in Definition 2.7 does not affect essentially the structural properties of G that we will investigate in the sequel. From now on, whenever we speak about δ_u and δ^*_u, we assume that these arbitrary choices of V_i and V_j have been already made.

Definition 2.8. Let G be a trapezoid graph and u be a vertex of G. The vertices of $N(u)$ are partitioned into four possibly empty sets:

(1) $N_0(u)$: vertices not adjacent to either δ_u or δ^*_u.
(2) $N_1(u)$: vertices adjacent to δ_u but not to δ^*_u.
(3) $N_2(u)$: vertices adjacent to δ^*_u but not to δ_u.
(4) $N_{12}(u)$: vertices adjacent to both δ_u and δ^*_u.

[†] In Observation 3.1(5) of [6], it is claimed that for an arbitrary trapezoid representation R of a connected trapezoid graph G, where V_i is a master component of u such that $D^*_u(V_i) \neq \emptyset$ and $R(V_i) \ll_R T_u$, it holds $R(D_u(V_i)) \ll_R T_u \ll_R R(D^*_u(V_i))$. However, the first part of the latter inequality is not true. For instance, in the trapezoid graph G of Figure 1, $V_2 = \{v_2\}$ is a master component of u, where $D^*_u(V_2) = \{V_3\} = \{\{v_3\}\} \neq \emptyset$ and $R(V_2) \ll_R T_u$. However, $V_4 = \{v_4\} \in D_u(V_2)$ and $T_u \ll_R T_{v_4}$, and thus, $R(D_u(V_2)) \not\ll_R T_u$.

In the following definition we partition the neighbors of a vertex of a trapezoid graph G into four possibly empty sets. Note that these sets depend on a given trapezoid representation R of G, in contrast to the four sets of Definition 2.8 that depend only on the graph G itself.

Definition 2.9. Let G be a trapezoid graph, R be a representation of G, and u be a vertex of G. Denote by $D_1(u,R)$ and $D_2(u,R)$ the sets of trapezoids of R that lie completely to the left and to the right of T_u in R, respectively. Then, the vertices of $N(u)$ are partitioned into four possibly empty sets:

(1) $N_0(u,R)$: vertices not adjacent to either $D_1(u,R)$ or $D_2(u,R)$.
(2) $N_1(u,R)$: vertices adjacent to $D_1(u,R)$ but not to $D_2(u,R)$.
(3) $N_2(u,R)$: vertices adjacent to $D_2(u,R)$ but not to $D_1(u,R)$.
(4) $N_{12}(u,R)$: vertices adjacent to both $D_1(u,R)$ and $D_2(u,R)$.

Suppose now that $\delta_u^* \neq \emptyset$, and let V_i be the master component of u that corresponds to δ_u, cf. Definition 2.7. Then, given any trapezoid representation R of G, we may assume w.l.o.g. that $R(V_i) \ll_R T_u$, by possibly performing a vertical axis flipping of R. The following lemma connects Definitions 2.8 and 2.9; in particular, it states that, if $R(V_i) \ll_R T_u$, then the partitions of the set $N(u)$ defined in these definitions coincide. This lemma will enable us to use in the vertex splitting (cf. Definition 2.11) the partition of the set $N(u)$ defined in Definition 2.8, independently of any trapezoid representation R of G, and regardless of any particular connected components V_i and V_j of $G \setminus N[u]$.

Lemma 2.10. *Let G be a trapezoid graph, R be a representation of G, and u be a vertex of G with $\delta_u^* \neq \emptyset$. Let V_i be the master component of u that corresponds to δ_u. If $R(V_i) \ll_R T_u$, then $N_X(u) = N_X(u,R)$ for every $X \in \{0,1,2,12\}$.*

2.3. A splitting algorithm

We define now the splitting of a vertex u of a trapezoid graph G, where $\delta_u^* \neq \emptyset$. Note that this splitting operation does not depend on any trapezoid representation of G. Intuitively, if the graph G was given along with a specific trapezoid representation R, this would have meant that we replace the trapezoid T_u in R by its two lines $l(T_u)$ and $r(T_u)$.

Definition 2.11. Let G be a trapezoid graph and u be a vertex of G, where $\delta_u^* \neq \emptyset$. The graph $G^{\#}(u)$ obtained by the *vertex splitting* of u is defined as follows:

(1) $V(G^{\#}(u)) = V(G) \setminus \{u\} \cup \{u_1, u_2\}$, where u_1 and u_2 are the two new vertices.
(2) $E(G^{\#}(u)) = E[V(G) \setminus \{u\}] \cup \{u_1x \mid x \in N_1(u)\} \cup \{u_2x \mid x \in N_2(u)\} \cup \{u_1x, u_2x \mid x \in N_{12}(u)\}$.

The vertices u_1 and u_2 are the *derivatives* of vertex u.

We state now the notion of a standard trapezoid representation with respect to a particular vertex.

Definition 2.12. Let G be a trapezoid graph and u be a vertex of G, where $\delta_u^* \neq \emptyset$. A trapezoid representation R of G is *standard with respect to u*, if the following properties are satisfied:

(1) $l(T_u) \ll_R R(N_0(u) \cup N_2(u))$.
(2) $R(N_0(u) \cup N_1(u)) \ll_R r(T_u)$.

Algorithm 1 Split-U

```
Input: A trapezoid graph $G$ and a vertex subset $U = \{u_1, u_2, \ldots, u_k\}$, such that $\delta^*_{u_i} \neq \emptyset$
    for all $i = 1, 2, \ldots, k$
Output: The permutation graph $G^{\#}(U)$
  $\overline{U} \leftarrow V(G) \setminus U$; $H_0 \leftarrow G$
  for $i = 1$ to $k$ do
    $H_i \leftarrow H^{\#}_{i-1}(u_i)$ {$H_i$ is obtained by the vertex splitting of $u_i$ in $H_{i-1}$}
  $G^{\#}(U) \leftarrow H_k[V(H_k) \setminus \overline{U}]$ {remove from $H_k$ all unsplitted vertices}
  return $G^{\#}(U)$
```

Now, given a trapezoid graph G and a vertex subset $U = \{u_1, u_2, \ldots, u_k\}$, such that $\delta^*_{u_i} \neq \emptyset$ for every $i = 1, 2, \ldots, k$, Algorithm Split-U returns a graph $G^{\#}(U)$ by splitting every vertex of U exactly once. At every step, Algorithm Split-U splits a vertex of U, and finally, it removes all vertices of the set $V(G) \setminus U$, which have not been split.

Remark 2.13. As mentioned in Remark 2.5, a similar algorithm, called Split-All, was presented in [6]. We would like to emphasize here the following four differences between the two algorithms. First, that Split-All gets as input a sibling-free graph G (two vertices u, v of a graph G are called *siblings*, if $N[u] = N[v]$; G is called *sibling-free* if G has no pair of sibling vertices), while our Algorithm Split-U gets as an input any graph (though, we will use it only for trapezoid graphs), which may contain also pairs of sibling vertices. Second, Split-All splits all the vertices of the input graph, while Split-U splits only a subset of them, which satisfy a special property. Third, the order of vertices that are split by Split-All depends on a certain property (inclusion-minimal neighbor set), while Split-U splits the vertices in an arbitrary order. Last, the main difference between these two algorithms is that they perform a different vertex splitting operation at every step, since Definitions 2.7 and 2.8 do not comply with the corresponding Definitions 4.1 and 4.2 of [6].

Theorem 2.14. *Let G be a trapezoid graph and $U = \{u_1, u_2, \ldots, u_k\}$ be a vertex subset of G, such that $\delta^*_{u_i} \neq \emptyset$ for every $i = 1, 2, \ldots, k$. Then, the graph $G^{\#}(U)$ obtained by Algorithm Split-U, is a permutation graph with $2k$ vertices. Furthermore, if G is acyclic, then $G^{\#}(U)$ is acyclic with respect to $\{u^1_i, u^2_i\}^k_{i=1}$, where u^1_i and u^2_i are the derivatives of u_i, $i = 1, 2, \ldots, k$.*

3. The recognition of bounded tolerance graphs

In this section we provide a reduction from the *monotone-Not-All-Equal-3-SAT (monotone-NAE-3-SAT)* problem to the problem of recognizing whether a given graph is a bounded tolerance graph. The problem of deciding whether a given monotone 3-CNF formula ϕ is NAE-satisfiable is known to be NP-complete. We can assume w.l.o.g. that each clause has three distinct literals, i.e. variables. Given a monotone 3-CNF formula ϕ, we construct in polynomial time a trapezoid graph H_ϕ, such that H_ϕ is a bounded tolerance graph if and only if ϕ is NAE-satisfiable. To this end, we construct first a permutation graph P_ϕ and a trapezoid graph G_ϕ.

3.1. The permutation graph P_ϕ

Consider a monotone 3-CNF formula $\phi = \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_k$ with k clauses and n boolean variables $x_1, x_2, \ldots, x_n$, such that $\alpha_i = (x_{r_{i,1}} \vee x_{r_{i,2}} \vee x_{r_{i,3}})$ for $i = 1, 2, \ldots, k$, where $1 \leq r_{i,1} < r_{i,2} < r_{i,3} \leq n$. We construct the permutation graph P_ϕ, along with a permutation representation R_P of P_ϕ, as follows. Let L_1 and L_2 be two parallel lines and let $\theta(\ell)$ denote the angle of the line ℓ with L_2 in R_P. For every clause α_i, $i = 1, 2, \ldots, k$, we correspond to each of the literals, i.e. variables, $x_{r_{i,1}}$, $x_{r_{i,2}}$, and $x_{r_{i,3}}$ a pair of intersecting lines with endpoints on L_1 and L_2. Namely, we correspond to the variable $x_{r_{i,1}}$ the pair $\{a_i, c_i\}$, to $x_{r_{i,2}}$ the pair $\{e_i, b_i\}$ and to $x_{r_{i,3}}$ the pair $\{d_i, f_i\}$, respectively, such that $\theta(a_i) > \theta(c_i)$, $\theta(e_i) > \theta(b_i)$, $\theta(d_i) > \theta(f_i)$, and such that the lines a_i, c_i lie completely to the left of e_i, b_i in R_P, and e_i, b_i lie completely to the left of d_i, f_i in R_P, as it is illustrated in Figure 2. Denote the lines that correspond to the variable $x_{r_{i,j}}$, $j = 1, 2, 3$, by $\ell^1_{i,j}$ and $\ell^2_{i,j}$, respectively, such that $\theta(\ell^1_{i,j}) > \theta(\ell^2_{i,j})$. That is, $(\ell^1_{i,1}, \ell^2_{i,1}) = (a_i, c_i)$, $(\ell^1_{i,2}, \ell^2_{i,2}) = (e_i, b_i)$, and $(\ell^1_{i,3}, \ell^2_{i,3}) = (d_i, f_i)$. Note that no line of a pair $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ intersects with a line of another pair $\{\ell^1_{i',j'}, \ell^2_{i',j'}\}$.

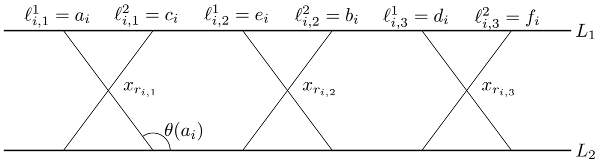


Figure 2: The six lines of the permutation graph P_ϕ, which correspond to the clause $\alpha_i = (x_{r_{i,1}} \vee x_{r_{i,2}} \vee x_{r_{i,3}})$ of the boolean formula ϕ.

Denote by S_p, $p = 1, 2, \ldots, n$, the set of pairs $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ that correspond to the variable x_p, i.e. $r_{i,j} = p$. We order the pairs $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ such that any pair of S_{p_1} lies completely to the left of any pair of S_{p_2}, whenever $p_1 < p_2$, while the pairs that belong to the same set S_p are ordered arbitrarily. For two consecutive pairs $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ and $\{\ell^1_{i',j'}, \ell^2_{i',j'}\}$ in S_p, where $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ lies to the left of $\{\ell^1_{i',j'}, \ell^2_{i',j'}\}$, we add a pair $\{u^{i',j'}_{i,j}, v^{i',j'}_{i,j}\}$ of parallel lines that intersect both $\ell^1_{i,j}$ and $\ell^1_{i',j'}$, but no other line. Note that $\theta(\ell^1_{i,j}) > \theta(u^{i',j'}_{i,j})$ and $\theta(\ell^1_{i',j'}) > \theta(u^{i',j'}_{i,j})$, while $\theta(u^{i',j'}_{i,j}) = \theta(v^{i',j'}_{i,j})$. This completes the construction. Denote the resulting permutation graph by P_ϕ, and the corresponding permutation representation of P_ϕ by R_P. Observe that P_ϕ has n connected components, which are called *blocks*, one for each variable $x_1, x_2, \ldots, x_n$.

An example of the construction of P_ϕ and R_P from ϕ with $k = 3$ clauses and $n = 4$ variables is illustrated in Figure 3. In this figure, the lines $u^{i',j'}_{i,j}$ and $v^{i',j'}_{i,j}$ are drawn in bold.

The formula ϕ has $3k$ literals, and thus the permutation graph P_ϕ has $6k$ lines $\ell^1_{i,j}, \ell^2_{i,j}$ in R_P, one pair for each literal. Furthermore, two lines $u^{i',j'}_{i,j}, v^{i',j'}_{i,j}$ correspond to each pair of consecutive pairs $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ and $\{\ell^1_{i',j'}, \ell^2_{i',j'}\}$ in R_P, except for the case where these pairs of lines belong to different variables, i.e. when $r_{i,j} \neq r_{i',j'}$. Therefore, since ϕ has n variables, there are $2(3k - n) = 6k - 2n$ lines $u^{i',j'}_{i,j}, v^{i',j'}_{i,j}$ in R_P. Thus, R_P has in total $12k - 2n$ lines, i.e. P_ϕ has $12k - 2n$ vertices. In the example of Figure 3, $k = 3$, $n = 4$, and thus, P_ϕ has 28 vertices.

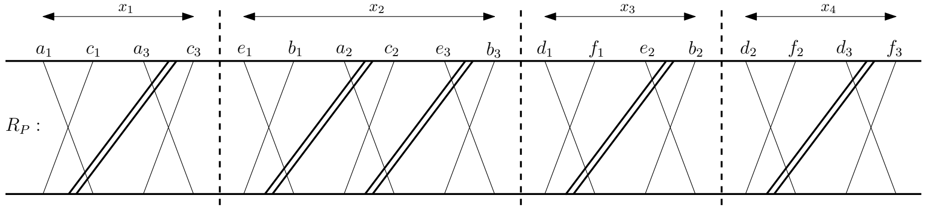


Figure 3: The permutation representation R_P of the permutation graph P_ϕ for $\phi = \alpha_1 \wedge \alpha_2 \wedge \alpha_3 = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4)$.

Let $m = 6k - n$, where $2m$ is the number of vertices in P_ϕ. We group the lines of R_P, i.e. the vertices of P_ϕ, into pairs $\{u_i^1, u_i^2\}_{i=1}^m$, as follows. For every clause α_i, $i = 1, 2, \ldots, k$, we group the lines $a_i, b_i, c_i, d_i, e_i, f_i$ into the three pairs $\{a_i, b_i\}$, $\{c_i, d_i\}$, and $\{e_i, f_i\}$. The remaining lines are grouped naturally according to the construction; namely, every two lines $\{u_{i,j}^{i',j'}, v_{i,j}^{i',j'}\}$ constitute a pair.

Lemma 3.1. *If the permutation graph P_ϕ is acyclic with respect to $\{u_i^1, u_i^2\}_{i=1}^m$ then the formula ϕ is NAE-satisfiable.*

The truth assignment $(x_1, x_2, x_3, x_4) = (1, 1, 0, 0)$ is NAE-satisfying for the formula ϕ of Figure 3. The acyclic permutation representation R_0 of P_ϕ with respect to $\{u_i^1, u_i^2\}_{i=1}^m$, which corresponds to this assignment, can be obtained from R_P by performing a horizontal axis flipping of the two blocks that correspond to the variables x_3 and x_4, respectively.

3.2. The trapezoid graphs G_ϕ and H_ϕ

Let $\{u_i^1, u_i^2\}_{i=1}^m$ be the pairs of vertices in the permutation graph P_ϕ and R_P be its permutation representation. We construct now from P_ϕ the trapezoid graph G_ϕ with m vertices $\{u_1, u_2, \ldots, u_m\}$, as follows. We replace in the permutation representation R_P for every $i = 1, 2, \ldots, m$ the lines u_i^1 and u_i^2 by the trapezoid T_{u_i}, which has u_i^1 and u_i^2 as its left and right lines, respectively. Let R_G be the resulting trapezoid representation of G_ϕ.

Finally, we construct from G_ϕ the trapezoid graph H_ϕ with $7m$ vertices, by adding to every trapezoid T_{u_i}, $i = 1, 2, \ldots, m$, six parallelograms $T_{u_{i,1}}, T_{u_{i,2}}, \ldots, T_{u_{i,6}}$ in the trapezoid representation R_G, as follows. Let ε be the smallest distance in R_G between two different endpoints on L_1, or on L_2. The right (resp. left) line of $T_{u_{1,1}}$ lies to the right (resp. left) of u_1^1, and it is parallel to it at distance $\frac{\varepsilon}{2}$. The right (resp. left) line of $T_{u_{1,2}}$ lies to the left of u_1^1, and it is parallel to it at distance $\frac{\varepsilon}{4}$ (resp. $\frac{3\varepsilon}{4}$). Moreover, the right (resp. left) line of $T_{u_{1,3}}$ lies to the left of u_1^1, and it is parallel to it at distance $\frac{3\varepsilon}{8}$ (resp. $\frac{7\varepsilon}{8}$). Similarly, the left (resp. right) line of $T_{u_{1,4}}$ lies to the left (resp. right) of u_1^2, and it is parallel to it at distance $\frac{\varepsilon}{2}$. The left (resp. right) line of $T_{u_{1,5}}$ lies to the right of u_1^2, and it is parallel to it at distance $\frac{\varepsilon}{4}$ (resp. $\frac{3\varepsilon}{4}$). Finally, the right (resp. left) line of $T_{u_{1,6}}$ lies to the right of u_1^2, and it is parallel to it at distance $\frac{3\varepsilon}{8}$ (resp. $\frac{7\varepsilon}{8}$), as illustrated in Figure 4.

After adding the parallelograms $T_{u_{1,1}}, T_{u_{1,2}}, \ldots, T_{u_{1,6}}$ to a trapezoid T_{u_1}, we update the smallest distance ε between two different endpoints on L_1, or on L_2 in the resulting representation, and we continue the construction iteratively for all $i = 2, \ldots, m$. Denote by H_ϕ the resulting trapezoid graph with $7m$ vertices, and by R_H the corresponding trapezoid representation. Note that in R_H, between the endpoints of the parallelograms $T_{u_{i,1}}$, $T_{u_{i,2}}$,

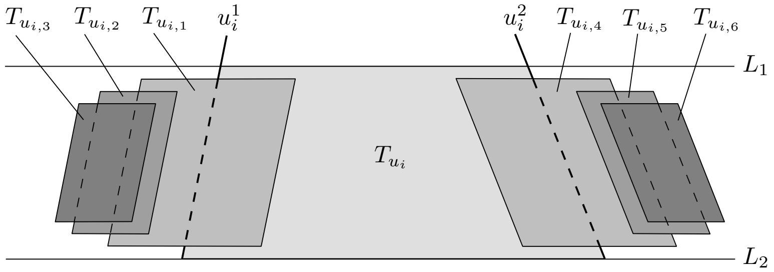


Figure 4: The addition of the six parallelograms $T_{u_{i,1}}, T_{u_{i,2}}, \ldots, T_{u_{i,6}}$ to the trapezoid T_{u_i}, $i = 1, 2, \ldots, m$, in the construction of the trapezoid graph H_ϕ from G_ϕ.

and $T_{u_{i,3}}$ (resp. $T_{u_{i,4}}$, $T_{u_{i,5}}$, and $T_{u_{i,6}}$) on L_1 and L_2, there are no other endpoints of H_ϕ, except those of u_i^1 (resp. u_i^2), for every $i = 1, 2, \ldots, m$. Furthermore, note that R_H is standard with respect to u_i, for every $i = 1, 2, \ldots, m$.

Theorem 3.2. *The formula ϕ is NAE-satisfiable if and only if the trapezoid graph H_ϕ is a bounded tolerance graph.*

For the sufficiency part of the proof of Theorem 3.2, the algorithm Split-All plays a crucial role. Namely, given the parallelogram graph H_ϕ (which is acyclic trapezoid by Lemma 2.3), we construct with this algorithm the acyclic permutation graph P_ϕ and then a NAE-satisfying assignment of the formula ϕ. Since monotone-NAE-3-SAT is NP-complete, the problem of recognizing bounded tolerance graphs is NP-hard by Theorem 3.2. Moreover, since this problem lies in NP [15], we summarize our results as follows.

Theorem 3.3. *Given a graph G, it is NP-complete to decide whether it is a bounded tolerance graph.*

4. The recognition of tolerance graphs

In this section we show that the reduction from the monotone-NAE-3-SAT problem to the problem of recognizing bounded tolerance graphs presented in Section 3, can be extended to the problem of recognizing general tolerance graphs. In particular, we prove that the constructed trapezoid graph H_ϕ is a tolerance graph if and only if it is a bounded tolerance graph. Then, the main result of this section follows.

Theorem 4.1. *Given a graph G, it is NP-complete to decide whether it is a tolerance graph. The problem remains NP-complete even if the given graph G is known to be a trapezoid graph.*

5. Concluding remarks

In this article we proved that both tolerance and bounded tolerance graph recognition problems are NP-complete, by providing a reduction from the monotone-NAE-3-SAT problem, thus answering a longstanding open question. The recognition of unit and of proper tolerance graphs, as well as of any other subclass of tolerance graphs, except bounded tolerance and bipartite tolerance graphs [5], remain interesting open problems [14].

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[2] K. P. Bogart, P. C. Fishburn, G. Isaak, and L. Langley. Proper and unit tolerance graphs. *Discrete Applied Mathematics*, 60(1-3):99–117, 1995.

[3] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph classes: a survey.* SIAM Monographs on Discrete Mathematics and Applications, 1999.

[4] A. H. Busch. A characterization of triangle-free tolerance graphs. *Discrete Applied Mathematics*, 154(3):471–477, 2006.

[5] A. H. Busch and G. Isaak. Recognizing bipartite tolerance graphs in linear time. In *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 12–20, 2007.

[6] F. Cheah and D. G. Corneil. On the structure of trapezoid graphs. *Discrete Applied Mathematics*, 66(2):109–133, 1996.

[7] F. Cheah and D. G. Corneil, 2009. Personal communication.

[8] S. Felsner. Tolerance graphs and orders. *Journal of Graph Theory*, 28:129–140, 1998.

[9] P. C. Fishburn and W. Trotter. Split semiorders. *Discrete Mathematics*, 195:111–126, 1999.

[10] M. C. Golumbic. *Algorithmic graph theory and perfect graphs (Annals of Discrete Mathematics, Vol. 57)*. North-Holland Publishing Co., 2nd edition, 2004.

[11] M. C. Golumbic and C. L. Monma. A generalization of interval graphs with tolerances. In *Proceedings of the 13th Southeastern Conference on Combinatorics, Graph Theory and Computing, Congressus Numerantium 35*, pages 321–331, 1982.

[12] M. C. Golumbic, C. L. Monma, and W. T. Trotter. Tolerance graphs. *Discrete Applied Mathematics*, 9(2):157–170, 1984.

[13] M. C. Golumbic and A. Siani. Coloring algorithms for tolerance graphs: reasoning and scheduling with interval constraints. In *Proceedings of the Joint International Conferences on Artificial Intelligence, Automated Reasoning, and Symbolic Computation (AISC/Calculemus)*, pages 196–207, 2002.

[14] M. C. Golumbic and A. N. Trenk. *Tolerance graphs.* Cambridge Studies in Advanced Mathematics, 2004.

[15] R. B. Hayward and R. Shamir. A note on tolerance graph recognition. *Discrete Applied Mathematics*, 143(1-3):307–311, 2004.

[16] G. Isaak, K. L. Nyman, and A. N. Trenk. A hierarchy of classes of bounded bitolerance orders. *Ars Combinatoria*, 69, 2003.

[17] M. Kaufmann, J. Kratochvíl, K. A. Lehmann, and A. R. Subramanian. Max-tolerance graphs as intersection graphs: cliques, cycles, and recognition. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 832–841, 2006.

[18] J. M. Keil and P. Belleville. Dominating the complements of bounded tolerance graphs and the complements of trapezoid graphs. *Discrete Applied Mathematics*, 140(1-3):73–89, 2004.

[19] L. Langley. *Interval tolerance orders and dimension*. PhD thesis, Dartmouth College, 1993.

[20] T.-H. Ma and J. P. Spinrad. On the 2-chain subgraph cover and related problems. *Journal of Algorithms*, 17(2):251–268, 1994.

[21] G. B. Mertzios and D. G. Corneil. Vertex splitting and the recognition of trapezoid graphs. Technical Report AIB-2009-16, Department of Computer Science, RWTH Aachen University, September 2009.

[22] G. B. Mertzios, I. Sau, and S. Zaks. A new intersection model and improved algorithms for tolerance graphs. *SIAM Journal on Discrete Mathematics*, 23(4):1800–1813, 2009.

[23] G. B. Mertzios, I. Sau, and S. Zaks. The recognition of tolerance and bounded tolerance graphs is NP-complete. Technical Report AIB-2009-06, Department of Computer Science, RWTH Aachen University, April 2009.

[24] G. Narasimhan and R. Manber. Stability and chromatic number of tolerance graphs. *Discrete Applied Mathematics*, 36:47–56, 1992.

[25] S. P. Ryan. Trapezoid order classification. *Order*, 15:341–354, 1998.

[26] J. P. Spinrad. *Efficient graph representations*, volume 19 of *Fields Institute Monographs.* American Mathematical Society, 2003.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 597-608
www.stacs-conf.org

DECIDABILITY OF THE INTERVAL TEMPORAL LOGIC $\mathsf{AB\bar{B}}$ OVER THE NATURAL NUMBERS

ANGELO MONTANARI[1] AND GABRIELE PUPPIS[2] AND PIETRO SALA[1] AND GUIDO SCIAVICCO[3]

[1] Department of Mathematics and Computer Science, Udine University, Italy
E-mail address: {angelo.montanari|pietro.sala}@dimi.uniud.it

[2] Computing Laboratory, Oxford University, England
E-mail address: Gabriele.Puppis@comlab.ox.ac.uk

[3] Department of Information, Engineering and Communications, Murcia University, Spain
E-mail address: guido@um.es

Abstract. In this paper, we focus our attention on the interval temporal logic of the Allen's relations "meets", "begins", and "begun by" ($\mathsf{AB\bar{B}}$ for short), interpreted over natural numbers. We first introduce the logic and we show that it is expressive enough to model distinctive interval properties, such as accomplishment conditions, to capture basic modalities of point-based temporal logic, such as the until operator, and to encode relevant metric constraints. Then, we prove that the satisfiability problem for $\mathsf{AB\bar{B}}$ over natural numbers is decidable by providing a small model theorem based on an original contraction method. Finally, we prove the EXPSPACE-completeness of the problem.

1. Introduction

Interval temporal logics are modal logics that allow one to represent and to reason about time intervals. It is well known that, on a linear ordering, one among thirteen different binary relations may hold between any pair of intervals, namely, "ends", "during", "begins", "overlaps", "meets", "before", together with their inverses, and the relation "equals" (the so-called Allen's relations [1])[1]. Allen's relations give rise to respective unary modal operators, thus defining the modal logic of time intervals HS introduced by Halpern and Shoham

1998 ACM Subject Classification: F.3: logics and meaning of programs; F.4: mathematical logic and formal languages.

Key words and phrases: interval temporal logics, compass structures, decidability, complexity.

The work has been partially supported by the GNCS project: "Logics, automata, and games for the formal verification of complex systems". Guido Sciavicco has also been supported by the Spain/South Africa Integrated Action N. HS2008-0006 on: "Metric interval temporal logics: Theory and Applications".

[1] We do not consider here the case of ternary relations. Amongst the multitude of *ternary* relations among intervals there is one of particular importance, which corresponds to the binary operation of concatenation of meeting intervals. The logic of such a ternary interval relation has been investigated by Venema in [20]. A systematic analysis of its fragments has been recently given by Hodkinson et al. [13].

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2488

in [12]. Some of these modal operators are actually definable in terms of others; in particular, if singleton intervals are included in the structure, it suffices to choose as basic the modalities corresponding to the relations "begins" B and "ends" E, and their transposes $\bar{\mathsf{B}}$, $\bar{\mathsf{E}}$. HS turns out to be highly undecidable under very weak assumptions on the class of interval structures over which its formulas are interpreted [12]. In particular, undecidability holds for any class of interval structures over linear orderings that contains at least one linear ordering with an infinite ascending or descending chain, thus including the natural time flows $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$. Undecidability of HS over finite structures directly follows from results in [15]. In [14], Lodaya sharpens the undecidability of HS showing that the two modalities B, E suffice for undecidability over dense linear orderings (in fact, the result applies to the class of all linear orderings [11]). Even though HS is very natural and the meaning of its operators is quite intuitive, for a long time such sweeping undecidability results have discouraged the search for practical applications and further investigations in the field. A renewed interest in interval temporal logics has been recently stimulated by the identification of some decidable fragments of HS, whose decidability does not depend on simplifying semantic assumptions such as locality and homogeneity [11]. This is the case with the fragments $\mathsf{B}\bar{\mathsf{B}}$, $\mathsf{E}\bar{\mathsf{E}}$ (logics of the "begins/begun by" and "ends/ended by" relations) [11], A, $\mathsf{A}\bar{\mathsf{A}}$ (logics of temporal neighborhood, whose modalities capture the "meets/met by" relations [10]), and D, $\mathsf{D}\bar{\mathsf{D}}$ (logics of the subinterval/superinterval relations) [3, 16].

In this paper, we focus our attention on the product logic $\mathsf{A}\mathsf{B}\bar{\mathsf{B}}$, obtained from the join of $\mathsf{B}\bar{\mathsf{B}}$ and A (the case of $\bar{\mathsf{A}}\mathsf{E}\bar{\mathsf{E}}$ is fully symmetric), interpreted over the linear order $\mathbb{N}$ of the natural numbers (or a finite prefix of it). The decidability of $\mathsf{B}\bar{\mathsf{B}}$ can be proved by translating it into the point-based propositional temporal logic of linear time with temporal modalities F (sometime in the future) and P (sometime in the past), which has the finite (pseudo-)model property and is decidable, e.g., [9]. In general, such a reduction to point-based temporal logics does not work: formulas of interval temporal logics are evaluated over pairs of points and translate into binary relations. For instance, this is the case with A. Unlike the case of $\mathsf{B}\bar{\mathsf{B}}$, when dealing with A one cannot abstract way from the left endpoint of intervals, as contradictory formulas may hold over intervals with the same right endpoint and a different left endpoint. The decidability of $\mathsf{A}\bar{\mathsf{A}}$, and thus that of its fragment A, over various classes of linear orderings has been proved by Bresolin et al. by reducing its satisfiability problem to that of the two-variable fragment of first-order logic over the same classes of structures [4], whose decidability has been proved by Otto in [18]. Optimal tableau methods for A with respect to various classes of interval structures can be found in [6, 7]. A decidable metric extension of A over the natural numbers has been proposed in [8]. A number of undecidable extensions of A, and $\mathsf{A}\bar{\mathsf{A}}$, have been given in [2, 5].

$\mathsf{A}\mathsf{B}\bar{\mathsf{B}}$ retains the simplicity of its constituents $\mathsf{B}\bar{\mathsf{B}}$ and A, but it improves a lot on their expressive power (as we shall show, such an increase in expressiveness is achieved at the cost of an increase in complexity). First, it allows one to express assertions that may be true at certain intervals, but at no subinterval of them, such as the conditions of accomplishment. Moreover, it makes it possible to easily encode the until operator of point-based temporal logic (this is possible neither with $\mathsf{B}\bar{\mathsf{B}}$ nor with A). Finally, meaningful metric constraints about the length of intervals can be expressed in $\mathsf{A}\mathsf{B}\bar{\mathsf{B}}$, that is, one can constrain an interval to be at least (resp., at most, exactly) k points long. We prove the decidability of $\mathsf{A}\mathsf{B}\bar{\mathsf{B}}$ interpreted over $\mathbb{N}$ by providing a small model theorem based on an original contraction method. To prove it, we take advantage of a natural (equivalent) interpretation of $\mathsf{A}\mathsf{B}\bar{\mathsf{B}}$ formulas over grid-like structures based on a bijection between the set of intervals over $\mathbb{N}$

and (a suitable subset of) the set of points of the $\mathbb{N} \times \mathbb{N}$ grid. In addition, we prove that the satisfiability problem for $\mathsf{AB\bar{B}}$ is EXPSPACE-complete (that for A is NEXPTIME-complete). In the proof of hardness, we use a reduction from the exponential-corridor tiling problem.

The paper is organized as follows. In Section 2 we introduce $\mathsf{AB\bar{B}}$. In Section 3, we prove the decidability of its satisfiability problem. We first describe the application of the contraction method to finite models and then we generalize it to infinite ones. In Section 4 we deal with computational complexity issues. Conclusions provide an assessment of the work and outline future research directions. Missing proofs can be found in [17].

2. The interval temporal logic $\mathsf{AB\bar{B}}$

In this section, we briefly introduce syntax and semantics of the logic $\mathsf{AB\bar{B}}$, which features three modal operators $\langle \mathsf{A} \rangle$, $\langle \mathsf{B} \rangle$, and $\langle \bar{\mathsf{B}} \rangle$ corresponding to the three Allen's relations A ("meets"), B ("begins"), and $\bar{\mathsf{B}}$ ("begun by"), respectively. We show that $\mathsf{AB\bar{B}}$ is expressive enough to capture the notion of accomplishment, to define the standard until operator of point-based temporal logics, and to encode metric conditions. Then, we introduce the basic notions of atom, type, and dependency. We conclude the section by providing an alternative interpretation of $\mathsf{AB\bar{B}}$ over labeled grid-like structures.

2.1. Syntax and semantics

Given a set $\mathcal{P}rop$ of propositional variables, formulas of $\mathsf{AB\bar{B}}$ are built up from $\mathcal{P}rop$ using the boolean connectives $\neg$ and $\vee$ and the unary modal operators $\langle \mathsf{A} \rangle$, $\langle \mathsf{B} \rangle$, $\langle \bar{\mathsf{B}} \rangle$. As usual, we shall take advantage of shorthands like $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $[\mathsf{A}]\varphi = \neg\langle \mathsf{A} \rangle \neg\varphi$, $[\mathsf{B}]\varphi = \neg\langle \mathsf{B} \rangle \neg\varphi$, $\top = p \vee \neg p$, and $\perp = p \wedge \neg p$, with $p \in \mathcal{P}rop$. Hereafter, we denote by $|\varphi|$ the size of φ.

We interpret formulas of $\mathsf{AB\bar{B}}$ in interval temporal structures over natural numbers endowed with the relations "meets", "begins", and "begun by". Precisely, we identify any given ordinal $N \leqslant \omega$ with the prefix of length N of the linear order of the natural numbers and we accordingly define $\mathbb{I}_N$ as the set of all non-singleton closed intervals $[x, y]$, with $x, y \in N$ and $x < y$. For any pair of intervals $[x, y], [x', y'] \in \mathbb{I}_N$, the Allen's relations "meets" A, "begins" B, and "begun by" $\bar{\mathsf{B}}$ are defined as follows (note that $\bar{\mathsf{B}}$ is the inverse relation of B):

- **"meets" relation:** $[x, y]\ \mathsf{A}\ [x', y']$ iff $y = x'$;
- **"begins" relation:** $[x, y]\ \mathsf{B}\ [x', y']$ iff $x = x'$ and $y' < y$;
- **"begun by" relation:** $[x, y]\ \bar{\mathsf{B}}\ [x', y']$ iff $x = x'$ and $y < y'$.

Given an *interval structure* $\mathcal{S} = (\mathbb{I}_N, \mathsf{A}, \mathsf{B}, \bar{\mathsf{B}}, \sigma)$, where $\sigma : \mathbb{I}_N \rightarrow \mathscr{P}(\mathcal{P}rop)$ is a labeling function that maps intervals in $\mathbb{I}_N$ to sets of propositional variables, and an initial interval I, we define the semantics of an $\mathsf{AB\bar{B}}$ formula as follows:

- $\mathcal{S}, I \vDash a$ iff $a \in \sigma(I)$, for any $a \in \mathcal{P}rop$;
- $\mathcal{S}, I \vDash \neg\varphi$ iff $\mathcal{S}, I \nvDash \varphi$;
- $\mathcal{S}, I \vDash \varphi_1 \vee \varphi_2$ iff $\mathcal{S}, I \vDash \varphi_1$ or $\mathcal{S}, I \vDash \varphi_2$;
- for every relation $R \in \{\mathsf{A}, \mathsf{B}, \bar{\mathsf{B}}\}$, $\mathcal{S}, I \vDash \langle R \rangle \varphi$ iff there is an interval $J \in \mathbb{I}_N$ such that $I\ R\ J$ and $\mathcal{S}, J \vDash \varphi$.

Given an interval structure $\mathcal{S}$ and a formula φ, we say that $\mathcal{S}$ *satisfies* φ if there is an interval I in $\mathcal{S}$ such that $\mathcal{S}, \mathrm{I} \models \varphi$. We say that φ is *satisfiable* if there exists an interval structure that satisfies it. We define the *satisfiability problem* for $\mathsf{AB\bar{B}}$ as the problem of establishing whether a given $\mathsf{AB\bar{B}}$-formula φ is satisfiable.

We conclude the section with some examples that account for $\mathsf{AB\bar{B}}$ expressive power. The first one shows how to encode in $\mathsf{AB\bar{B}}$ conditions of accomplishment (think of formula φ as the assertion: "Mr. Jones flew from Venice to Nancy"): $\langle \mathsf{A} \rangle \big(\varphi \wedge [\mathsf{B}](\neg\varphi \wedge [\mathsf{A}]\neg\varphi) \wedge [\bar{\mathsf{B}}]\neg\varphi\big)$. Formulas of point-based temporal logics of the form $\psi \ \mathsf{U} \ \varphi$, using the standard until operator, can be encoded in $\mathsf{AB\bar{B}}$ (where atomic intervals are two-point intervals) as follows: $\langle \mathsf{A} \rangle \big([\mathsf{B}]\bot \wedge \varphi\big) \vee \langle \mathsf{A} \rangle \big(\langle \mathsf{A} \rangle([\mathsf{B}]\bot \wedge \varphi) \wedge [\mathsf{B}](\langle \mathsf{A} \rangle([\mathsf{B}]\bot \wedge \psi))\big)$. Finally, metric conditions like: "φ holds over a right neighbor interval of length greater than k (resp., less than k, equal to k)" can be captured by the following $\mathsf{AB\bar{B}}$ formula: $\langle \mathsf{A} \rangle \big(\varphi \wedge \langle \mathsf{B} \rangle^{\mathsf{k}} \top\big)$ (resp., $\langle \mathsf{A} \rangle \big(\varphi \wedge [\mathsf{B}]^{\mathsf{k}-1} \bot\big)$, $\langle \mathsf{A} \rangle \big(\varphi \wedge [\mathsf{B}]^{\mathsf{k}} \bot \wedge \langle \mathsf{B} \rangle^{\mathsf{k}-1} \top\big))$[2].

2.2. Atoms, types, and dependencies

Let $\mathcal{S} = (\mathbb{I}_{\mathsf{N}}, \mathsf{A}, \mathsf{B}, \bar{\mathsf{B}}, \sigma)$ be an interval structure and φ be a formula of $\mathsf{AB\bar{B}}$. In the sequel, we shall compare intervals in $\mathcal{S}$ with respect to the set of subformulas of φ they satisfy. To do that, we introduce the key notions of φ-atom, φ-type, φ-cluster, and φ-shading.

First of all, we define the *closure* $\mathcal{C}l(\varphi)$ of φ as the set of all subformulas of φ and of their negations (we identify $\neg\neg\alpha$ with α, $\neg\langle \mathsf{A} \rangle \alpha$ with $[\mathsf{A}]\neg\alpha$, etc.). For technical reasons, we also introduce the *extended closure* $\mathcal{C}l^{+}(\varphi)$, which is defined as the set of all formulas in $\mathcal{C}l(\varphi)$ plus all formulas of the forms $\langle \mathsf{R} \rangle \alpha$ and $\neg\langle \mathsf{R} \rangle \alpha$, with $\mathsf{R} \in \{\mathsf{A}, \mathsf{B}, \bar{\mathsf{B}}\}$ and $\alpha \in \mathcal{C}l(\varphi)$.

A φ*-atom* is any non-empty set $\mathsf{F} \subseteq \mathcal{C}l^{+}(\varphi)$ such that (i) for every $\alpha \in \mathcal{C}l^{+}(\varphi)$, we have $\alpha \in \mathsf{F}$ iff $\neg\alpha \notin \mathsf{F}$ and (ii) for every $\gamma = \alpha \vee \beta \in \mathcal{C}l^{+}(\varphi)$, we have $\gamma \in \mathsf{F}$ iff $\alpha \in \mathsf{F}$ or $\beta \in \mathsf{F}$ (intuitively, a φ*-atom* is a maximal *locally consistent* set of formulas chosen from $\mathcal{C}l^{+}(\varphi)$). Note that the cardinalities of both sets $\mathcal{C}l(\varphi)$ and $\mathcal{C}l^{+}(\varphi)$ are *linear* in the number $|\varphi|$ of subformulas of φ, while the number of φ-atoms is *at most exponential* in $|\varphi|$ (precisely, we have $|\mathcal{C}l(\varphi)| = 2|\varphi|$, $|\mathcal{C}l^{+}(\varphi)| = 14|\varphi|$, and there are at most $2^{7|\varphi|}$ distinct atoms).

We also associate with each interval $\mathrm{I} \in \mathcal{S}$ the set of all formulas $\alpha \in \mathcal{C}l^{+}(\varphi)$ such that $\mathcal{S}, \mathrm{I} \models \alpha$. Such a set is called φ*-type* of I and it is denoted by $\mathcal{T}ype_{\mathcal{S}}(\mathrm{I})$. We have that every φ-type is a φ-atom, but not vice versa. Hereafter, we shall omit the argument φ, thus calling a φ-atom (resp., a φ-type) simply an atom (resp., a type).

Given an atom F, we denote by $\mathcal{O}bs(\mathsf{F})$ the set of all *observables* of F, namely, the formulas $\alpha \in \mathcal{C}l(\varphi)$ such that $\alpha \in \mathsf{F}$. Similarly, given an atom F and a relation $\mathsf{R} \in \{\mathsf{A}, \mathsf{B}, \bar{\mathsf{B}}\}$, we denote by $\mathcal{R}eq_{\mathsf{R}}(\mathsf{F})$ the set of all R*-requests* of F, namely, the formulas $\alpha \in \mathcal{C}l(\varphi)$ such that $\langle \mathsf{R} \rangle \alpha \in \mathsf{F}$. Taking advantage of the above sets, we can define the following two relations between atoms F and G:

$$\mathsf{F} \xrightarrow{\mathsf{A}} \mathsf{G} \quad \text{iff} \quad \mathcal{R}eq_{\mathsf{A}}(\mathsf{F}) = \mathcal{O}bs(\mathsf{G}) \cup \mathcal{R}eq_{\mathsf{B}}(\mathsf{G}) \cup \mathcal{R}eq_{\bar{\mathsf{B}}}(\mathsf{G});$$

$$\mathsf{F} \xrightarrow{\mathsf{B}} \mathsf{G} \quad \text{iff} \quad \begin{cases} \mathcal{O}bs(\mathsf{F}) \cup \mathcal{R}eq_{\bar{\mathsf{B}}}(\mathsf{F}) \subseteq \mathcal{R}eq_{\bar{\mathsf{B}}}(\mathsf{G}) \subseteq \mathcal{O}bs(\mathsf{F}) \cup \mathcal{R}eq_{\bar{\mathsf{B}}}(\mathsf{F}) \cup \mathcal{R}eq_{\mathsf{B}}(\mathsf{F}), \\ \mathcal{O}bs(\mathsf{G}) \cup \mathcal{R}eq_{\mathsf{B}}(\mathsf{G}) \subseteq \mathcal{R}eq_{\mathsf{B}}(\mathsf{F}) \subseteq \mathcal{O}bs(\mathsf{G}) \cup \mathcal{R}eq_{\mathsf{B}}(\mathsf{G}) \cup \mathcal{R}eq_{\bar{\mathsf{B}}}(\mathsf{G}). \end{cases}$$

[2] It is not difficult to show that $\mathsf{AB\bar{B}}$ subsumes the metric extension of A given in [8]. A simple game-theoretic argument shows that the former is in fact strictly more expressive than the latter.

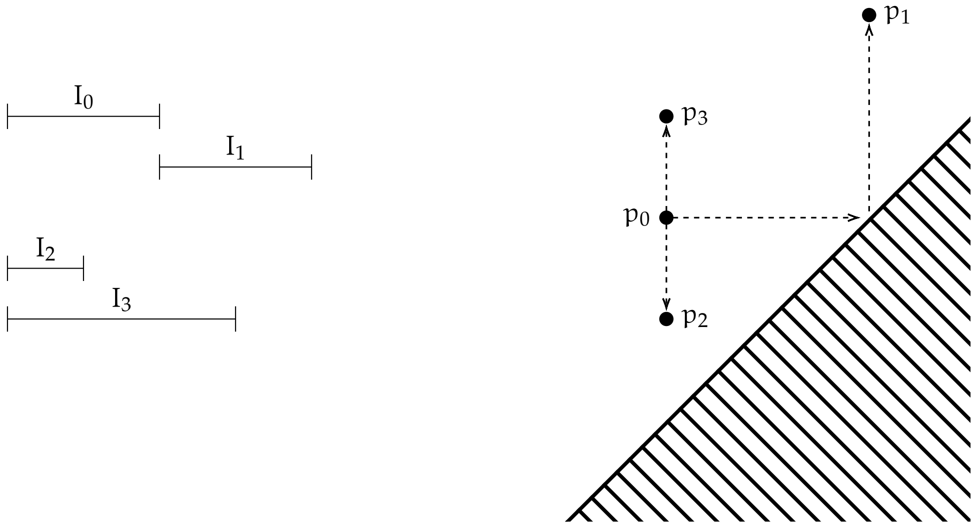


Figure 1: Correspondence between intervals and points of a discrete grid.

Note that the relation $\xrightarrow{B}$ is transitive, while $\xrightarrow{A}$ is not. Moreover, both $\xrightarrow{A}$ and $\xrightarrow{B}$ satisfy a *view-to-type dependency*, namely, for every pair of intervals I, J in $\mathcal{S}$, we have that

$$\begin{array}{lcl} I\ A\ J & \text{implies} & \mathcal{T}\mathit{ype}_{\mathcal{S}}(I) \xrightarrow{A} \mathcal{T}\mathit{ype}_{\mathcal{S}}(J) \\ I\ B\ J & \text{implies} & \mathcal{T}\mathit{ype}_{\mathcal{S}}(I) \xrightarrow{B} \mathcal{T}\mathit{ype}_{\mathcal{S}}(J). \end{array}$$

Relations $\xrightarrow{A}$ and $\xrightarrow{B}$ will come into play in the definition of consistency conditions (see Definition 2.1).

2.3. Compass structures

The logic $AB\bar{B}$ can be equivalently interpreted over grid-like structures (the so-called compass structures [20]) by exploiting the existence of a natural bijection between the intervals $I = [x, y]$ and the points $p = (x, y)$ of an $N \times N$ grid such that $x < y$. As an example, Figure 1 depicts four intervals $I_0, ..., I_3$ such that $I_0\ A\ I_1$, $I_0\ B\ I_2$, and $I_0\ \bar{B}\ I_3$, together with the corresponding points $p_0, ..., p_3$ of a discrete grid (note that the three Allen's relations $A, B, \bar{B}$ between intervals are mapped to corresponding spatial relations between points; for the sake of readability, we name the latter ones as the former ones).

Definition 2.1. Given an $AB\bar{B}$ formula φ, a (consistent and fulfilling) *compass* (φ-)*structure* of length $N \leqslant \omega$ is a pair $\mathcal{G} = (\mathbb{P}_N, \mathcal{L})$, where $\mathbb{P}_N$ is the set of points $p = (x, y)$, with $0 \leqslant x < y < N$, and $\mathcal{L}$ is function that maps any point $p \in \mathbb{P}_N$ to a (φ-)atom $\mathcal{L}(p)$ in such a way that

- for every pair of points $p, q \in \mathbb{P}_N$ and every relation $R \in \{A, B\}$, if $p\ R\ q$ holds, then $\mathcal{L}(p) \xrightarrow{R} \mathcal{L}(q)$ follows (**consistency**);
- for every point $p \in \mathbb{P}_N$, every relation $R \in \{A, B, \bar{B}\}$, and every formula $\alpha \in \mathcal{R}\mathit{eq}_R(\mathcal{L}(p))$, there is a point $q \in \mathbb{P}_N$ such that $p\ R\ q$ and $\alpha \in \mathcal{O}\mathit{bs}(\mathcal{L}(q))$ (**fulfillment**).

We say that a compass (φ-)structure $\mathcal{G} = (\mathbb{P}_N, \mathcal{L})$ *features* a formula α if there is a point $p \in \mathbb{P}_N$ such that $\alpha \in \mathcal{L}(p)$. The following proposition implies that the satisfiability problem for $\mathsf{AB\bar{B}}$ is reducible to the problem of deciding, for any given formula φ, whether there exists a φ-compass structure that features φ.

Proposition 2.2. *An $\mathsf{AB\bar{B}}$-formula φ is satisfied by some interval structure if and only if it is featured by some (φ-)compass structure.*

3. Deciding the satisfiability problem for $\mathsf{AB\bar{B}}$

In this section, we prove that the satisfiability problem for $\mathsf{AB\bar{B}}$ is decidable by providing a "small-model theorem" for the satisfiable formulas of the logic. For the sake of simplicity, we first show that the satisfiability problem for $\mathsf{AB\bar{B}}$ interpreted over *finite* interval structures is decidable and then we generalize such a result to all (finite or infinite) interval structures.

As a preliminary step, we introduce the key notion of shading. Let $\mathcal{G} = (\mathbb{P}_N, \mathcal{L})$ be a compass structure of length $N \leqslant \omega$ and let $0 \leqslant y < N$. The *shading of the row* y *of* $\mathcal{G}$ is the set $\mathcal{S}hading_{\mathcal{G}}(y) = \{\mathcal{L}(x, y) : 0 \leqslant x < y\}$, namely, the set of the atoms of all points in $\mathbb{P}_N$ whose vertical coordinate has value y (basically, we interpret different atoms as different colors). Clearly, for every pair of atoms F and F' in $\mathcal{S}hading_{\mathcal{G}}(y)$, we have $\mathcal{R}eq_A(F) = \mathcal{R}eq_A(F')$.

3.1. A small-model theorem for finite structures

Let φ be an $\mathsf{AB\bar{B}}$ formula. Let us assume that φ is featured by a *finite* compass structure $\mathcal{G} = (\mathbb{P}_N, \mathcal{L})$, with $N < \omega$. In fact, without loss of generality, we can assume that φ belongs to the atom associated with a point $p = (0, y)$ of $\mathcal{G}$, with $0 < y < N$. We prove that we can restrict our attention to compass structures $\mathcal{G} = (\mathbb{P}_N, \mathcal{L})$, where N is bounded by a double exponential in $|\varphi|$. We start with the following lemma that proves a simple, but crucial, property of the relations $\xrightarrow{A}$ and $\xrightarrow{B}$ (the proof can be found in [17]).

Lemma 3.1. *If $F \xrightarrow{A} H$ and $G \xrightarrow{B} H$ hold for some atoms F, G, H, then $F \xrightarrow{A} G$ holds as well.*

The next lemma shows that, under suitable conditions, a given compass structure $\mathcal{G}$ may be reduced in length, preserving the existence of atoms featuring φ.

Lemma 3.2. *Let $\mathcal{G}$ be a compass structure featuring φ. If there exist two rows $0 < y_0 < y_1 < N$ in $\mathcal{G}$ such that $\mathcal{S}hading_{\mathcal{G}}(y_0) \subseteq \mathcal{S}hading_{\mathcal{G}}(y_1)$, then there exists a compass structure $\mathcal{G}'$ of length $N' < N$ that features φ.*

Proof. Suppose that $0 < y_0 < y_1 < N$ are two rows of $\mathcal{G}$ such that $\mathcal{S}hading_{\mathcal{G}}(y_0) \subseteq \mathcal{S}hading_{\mathcal{G}}(y_1)$. Then, there is a function $f : \{0, ..., y_0 - 1\} \to \{0, ..., y_1 - 1\}$ such that, for every $0 \leqslant x < y_0$, $\mathcal{L}(x, y_0) = \mathcal{L}(f(x), y_1)$. Let $k = y_1 - y_0$, $N' = N - k$ ($< N$), and $\mathbb{P}_{N'}$ be the portion of the grid that consists of all points $p = (x, y)$, with $0 \leqslant x < y < N'$. We extend f to a function that maps points in $\mathbb{P}_{N'}$ to points in $\mathbb{P}_N$ as follows:

- if $p = (x, y)$, with $0 \leqslant x < y < y_0$, then we simply let $f(p) = p$;
- if $p = (x, y)$, with $0 \leqslant x < y_0 \leqslant y$, then we let $f(p) = (f(x), y + k)$;
- if $p = (x, y)$, with $y_0 \leqslant x < y$, then we let $f(p) = (x + k, y + k)$.

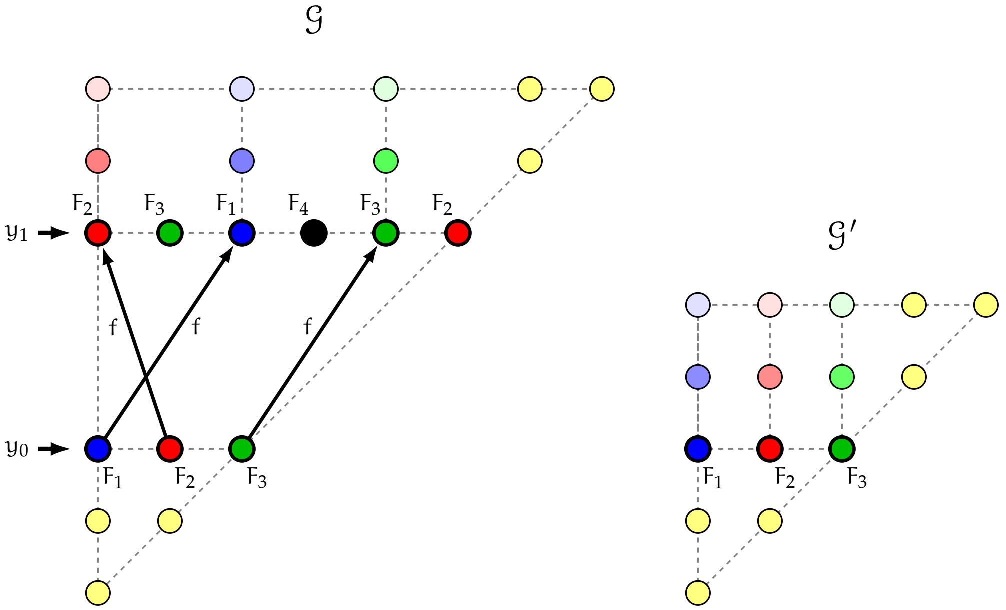


Figure 2: Contraction $\mathcal{G}'$ of a compass structure $\mathcal{G}$.

We denote by $\mathcal{L}'$ the labeling of $\mathbb{P}_{N'}$ such that, for every point $p \in \mathbb{P}_{N'}$, $\mathcal{L}'(p) = \mathcal{L}(f(p))$ and we denote by $\mathcal{G}'$ the resulting structure $(\mathbb{P}_{N'}, \mathcal{L}')$ (see Figure 2). We have to prove that $\mathcal{G}'$ is a *consistent* and *fulfilling* compass structure that features φ (see Definition 2.1). First, we show that $\mathcal{G}'$ satisfies the consistency conditions for the relations B and A; then we show that $\mathcal{G}'$ satisfies the fulfillment conditions for the $\bar{B}$-, B-, and A-requests; finally, we show that $\mathcal{G}'$ features φ.

Consistency with relation B. Consider two points $p = (x, y)$ and $p' = (x', y')$ in $\mathcal{G}'$ such that $p \; B \; p'$, i.e., $0 \leqslant x = x' < y' < y < N'$. We prove that $\mathcal{L}'(p) \xrightarrow{B} \mathcal{L}'(p')$ by distinguishing among the following three cases (note that exactly one of such cases holds):

(1) $y < y_0$ and $y' < y_0$,

(2) $y \geqslant y_0$ and $y' \geqslant y_0$,

(3) $y \geqslant y_0$ and $y' < y_0$.

If $y < y_0$ and $y' < y_0$, then, by construction, we have $f(p) = p$ and $f(p') = p'$. Since $\mathcal{G}$ is a (consistent) compass structure, we immediately obtain $\mathcal{L}'(p) = \mathcal{L}(p) \xrightarrow{B} \mathcal{L}(p') = \mathcal{L}'(p')$.
If $y \geqslant y_0$ and $y \geqslant y_0$, then, by construction, we have either $f(p) = (f(x), y + k)$ or $f(p) = (x + k, y + k)$, depending on whether $x < y_0$ or $x \geqslant y_0$. Similarly, we have either $f(p') = (f(x'), y' + k) = (f(x), y' + k)$ or $f(p') = (x' + k, y' + k) = (x + k, y' + k)$. This implies $f(p) \; B \; f(p')$ and thus, since $\mathcal{G}$ is a (consistent) compass structure, we have $\mathcal{L}'(p) = \mathcal{L}(f(p)) \xrightarrow{B} \mathcal{L}(f(p')) = \mathcal{L}'(p')$.
If $y \geqslant y_0$ and $y' < y_0$, then, since $x < y' < y_0$, we have by construction $f(p) = (f(x), y + k)$ and $f(p') = p'$. Moreover, if we consider the point $p'' = (x, y_0)$ in $\mathcal{G}'$, we easily see that (i) $f(p'') = (f(x), y_1)$, (ii) $f(p) \; B \; f(p'')$ (whence $\mathcal{L}(f(p)) \xrightarrow{B} \mathcal{L}(f(p''))$), (iii) $\mathcal{L}(f(p'')) = \mathcal{L}(p'')$, and (iv) $p'' \; B \; p'$ (whence $\mathcal{L}(p'') \xrightarrow{B} \mathcal{L}(p')$). It thus follows that $\mathcal{L}'(p) = \mathcal{L}(f(p)) \xrightarrow{B} \mathcal{L}(f(p'')) = \mathcal{L}(p'') \xrightarrow{B} \mathcal{L}(p') = \mathcal{L}(f(p')) = \mathcal{L}'(p')$. Finally, by exploiting the transitivity of the relation $\xrightarrow{B}$, we obtain $\mathcal{L}'(p) \xrightarrow{B} \mathcal{L}'(p')$.

Consistency with relation A. Consider two points $p = (x, y)$ and $p' = (x', y')$ such that $p \; A \; p'$, i.e., $0 \leqslant x < y = x' < y' < N'$. We define $p'' = (y, y + 1)$ in such a way that $p \; A \; p''$ and $p' \; B \; p''$ and we distinguish between the following two cases:

(1) $y \geqslant y_0$,

(2) $y < y_0$.

If $y \geqslant y_0$, then, by construction, we have $f(p) \; A \; f(p'')$. Since $\mathcal{G}$ is a (consistent) compass structure, it follows that $\mathcal{L}'(p) = \mathcal{L}(f(p)) \xrightarrow{A} \mathcal{L}(f(p'')) = \mathcal{L}'(p'')$.

If $y < y_0$, then, by construction, we have $\mathcal{L}(p'') = \mathcal{L}(f(p''))$. Again, since $\mathcal{G}$ is a (consistent) compass structure, it follows that $\mathcal{L}'(p) = \mathcal{L}(f(p)) = \mathcal{L}(p) \xrightarrow{A} \mathcal{L}(p'') = \mathcal{L}(f(p'')) = \mathcal{L}'(p'')$.

In both cases we have $\mathcal{L}'(p) \xrightarrow{A} \mathcal{L}'(p'')$. Now, we recall that $p' \; B \; p''$ and that, by previous arguments, $\mathcal{G}'$ is consistent with the relation B. We thus have $\mathcal{L}'(p') \xrightarrow{B} \mathcal{L}'(p'')$. Finally, by applying Lemma 3.1, we obtain $\mathcal{L}'(p) \xrightarrow{A} \mathcal{L}'(p')$.

Fulfillment of B-requests. Consider a point $p = (x, y)$ in $\mathcal{G}'$ and some B-request $\alpha \in \mathcal{R}eq_B\big(\mathcal{L}'(p)\big)$ associated with it. Since, by construction, $\alpha \in \mathcal{R}eq_B\big(\mathcal{L}(f(p))\big)$ and $\mathcal{G}$ is a (fulfilling) compass structure, we know that $\mathcal{G}$ contains a point $q' = (x', y')$ such that $f(p) \; B \; q'$ and $\alpha \in \mathcal{O}bs\big(\mathcal{L}(q')\big)$. We prove that $\mathcal{G}'$ contains a point p' such that $p \; B \; p'$ and $\alpha \in \mathcal{O}bs\big(\mathcal{L}'(p')\big)$ by distinguishing among the following three cases (note that exactly one of such cases holds):

(1) $y < y_0$

(2) $y' \geqslant y_1$,

(3) $y \geqslant y_0$ and $y' < y_1$.

If $y < y_0$, then, by construction, we have $p = f(p)$ and $q' = f(q')$. Therefore, we simply define $p' = q'$ in such a way that $p = f(p) \; B \; q' = p'$ and $\alpha \in \mathcal{O}bs\big(\mathcal{L}'(p')\big)$ $\big(= \mathcal{O}bs\big(\mathcal{L}(f(p'))\big) = \mathcal{O}bs\big(\mathcal{L}(q')\big)\big)$.

If $y' \geqslant y_1$, then, by construction, we have either $f(p) = (f(x), y+k)$ or $f(p) = (x+k, y+k)$, depending on whether $x < y_0$ or $x \geqslant y_0$. We define $p' = (x, y' - k)$ in such a way that $p \; B \; p'$. Moreover, we observe that either $f(p') = (f(x), y')$ or $f(p') = (x+k, y')$, depending on whether $x < y_0$ or $x \geqslant y_0$, and in both cases $f(p') = q'$ follows. This shows that $\alpha \in \mathcal{O}bs\big(\mathcal{L}'(p')\big)$ $\big(= \mathcal{O}bs\big(\mathcal{L}(f(p')\big) = \mathcal{O}bs\big(\mathcal{L}(q')\big)\big)$.

If $y \geqslant y_0$ and $y' < y_1$, then we define $\bar{p} = (x, y_0)$ and $\bar{q} = (x', y_1)$ and we observe that $f(p) \; B \; \bar{q}$, $\bar{q} \; B \; q'$, and $f(\bar{p}) = \bar{q}$. From $f(p) \; B \; \bar{q}$ and $\bar{q} \; B \; q'$, it follows that $\alpha \in \mathcal{R}eq_B\big(\mathcal{L}(\bar{q})\big)$ and hence $\alpha \in \mathcal{R}eq_B\big(\mathcal{L}(\bar{p})\big)$. Since $\mathcal{G}$ is a (fulfilling) compass structure, we know that there is a point p' such that $\bar{p} \; B \; p'$ and $\alpha \in \mathcal{O}bs\big(\mathcal{L}(\bar{p}')\big)$. Moreover, since $\bar{p} \; B \; p'$, we have $f(p') = p'$, from which we obtain $p \; B \; p'$ and $\alpha \in \mathcal{O}bs\big(\mathcal{L}(p')\big)$.

Fulfillment of $\bar{B}$-requests. The proof that $\mathcal{G}'$ fulfills all $\bar{B}$-requests of its atoms is symmetric with respect to the previous one.

Fulfillment of A-requests. Consider a point $p = (x, y)$ in $\mathcal{G}'$ and some A-request $\alpha \in \mathcal{R}eq_A\big(\mathcal{L}'(p)\big)$ associated with p in $\mathcal{G}'$. Since, by previous arguments, $\mathcal{G}'$ fulfills all $\bar{B}$-requests of its atoms, it is sufficient to prove that either $\alpha \in \mathcal{O}bs\big(\mathcal{L}'(p')\big)$ or $\alpha \in \mathcal{R}eq_{\bar{B}}\big(\mathcal{L}'(p')\big)$, where $p' = (y, y+1)$. This can be easily proved by distinguishing among the three cases $y < y_0 - 1$, $y = y_0 - 1$, and $y \geqslant y_0$.

Featured formulas. Recall that, by previous assumptions, $\mathcal{G}$ contains a point $p = (0, y)$, with $0 < y < N$, such that $\varphi \in \mathcal{L}(p)$. If $y \leqslant y_0$, then, by construction, we have

$\varphi \in \mathcal{L}'(p)$ $(= \mathcal{L}(f(p)) = \mathcal{L}(p))$. Otherwise, if $y > y_0$, we define $q = (0, y_0)$ and we observe that $q\ \bar{B}\ p$. Since $\mathcal{G}$ is a (consistent) compass structure and $\langle \bar{B} \rangle \varphi \in \mathcal{C}l^+(\varphi)$, we have that $\varphi \in \mathcal{R}eq_{\bar{B}}\big(\mathcal{L}(q)\big)$. Moreover, by construction, we have $\mathcal{L}'(q) = \mathcal{L}(f(q))$ and hence $\varphi \in \mathcal{R}eq_{\bar{B}}\big(\mathcal{L}'(q)\big)$. Finally, since $\mathcal{G}'$ is a (fulfilling) compass structure, we know that there is a point p' in $\mathcal{G}'$ such that $f(q)\ \bar{B}\ p'$ and $\varphi \in \mathcal{O}bs\big(\mathcal{L}'(p')\big)$. □

On the grounds of the above result, we can provide a suitable upper bound for the length of a minimal finite interval structure that satisfies φ, if there exists any. This yields a straightforward, but inefficient, 2EXPSPACE algorithm that decides whether a given $\mathsf{AB\bar{B}}$-formula φ is satisfiable over finite interval structures.

Theorem 3.3. *An $\mathsf{AB\bar{B}}$-formula φ is satisfied by some finite interval structure iff it is featured by some compass structure of length $N \leqslant 2^{2^{7|\varphi|}}$ (i.e., double exponential in $|\varphi|$).*

Proof. One direction is trivial. We prove the other one ("only if" part). Suppose that φ is satisfied by a finite interval structure $\mathcal{S}$. By Proposition 2.2, there is a compass structure $\mathcal{G}$ that features φ and has finite length $N < \omega$. Without loss of generality, we can assume that N is minimal among all finite compass structures that feature φ. We recall from Section 2.2 that $\mathcal{G}$ contains at most $2^{7|\varphi|}$ distinct atoms. This implies that there exist at most $2^{2^{7|\varphi|}}$ different shadings of the form $\mathcal{S}hading_{\mathcal{G}}(y)$, with $0 \leqslant y < N$. Finally, by applying Lemma 3.2, we obtain $N \leqslant 2^{2^{7|\varphi|}}$ (otherwise, there would exist two rows $0 < y_0 < y_1 < N$ such that $\mathcal{S}hading_{\mathcal{G}}(y_0) = \mathcal{S}hading_{\mathcal{G}}(y_1)$, which is against the hypothesis of minimality of N). □

3.2. A small-model theorem for infinite structures

In general, compass structures that feature φ may be infinite. Here, we prove that, without loss of generality, we can restrict our attention to sufficiently "regular" infinite compass structures, which can be represented in double exponential space with respect to $|\varphi|$. To do that, we introduce the notion of periodic compass structure.

Definition 3.4. An infinite compass structure $\mathcal{G} = (\mathbb{P}_\omega, \mathcal{L})$ is *periodic*, with *threshold* $\widetilde{y}_0$, *period* $\widetilde{y}$, and *binding* $\widetilde{g} : \{0, ..., \widetilde{y}_0 + \widetilde{y} - 1\} \rightarrow \{0, ..., \widetilde{y}_0 - 1\}$, if the following conditions are satisfied:

- for every $\widetilde{y}_0 + \widetilde{y} \leqslant x < y$, we have $\mathcal{L}(x, y) = \mathcal{L}(x - \widetilde{y}, y - \widetilde{y})$,
- for every $0 \leqslant x < \widetilde{y}_0 + \widetilde{y} \leqslant y$, we have $\mathcal{L}(x, y) = \mathcal{L}(\widetilde{g}(x), y - \widetilde{y})$.

Figure 3 gives an example of a periodic compass structure (the arrows represent some relationships between points induced by the binding function $\widetilde{g}$). Note that any periodic compass structure $\mathcal{G} = (\mathbb{P}_\omega, \mathcal{L})$ can be finitely represented by specifying (i) its threshold $\widetilde{y}_0$, (ii) its period $\widetilde{y}$, (iii) its binding $\widetilde{g}$, and (iv) the labeling $\mathcal{L}$ restricted to the portion $\mathbb{P}_{\widetilde{y}_0 + \widetilde{y} - 1}$ of the domain.

The following theorem leads immediately to a 2EXPSPACE algorithm that decides whether a given $\mathsf{AB\bar{B}}$-formula φ is satisfiable over infinite interval structures (the proof is provided in [17]).

Theorem 3.5. *An $\mathsf{AB\bar{B}}$-formula φ is satisfied by an infinite interval structure iff it is featured by a periodic compass structure with threshold $\widetilde{y}_0 < 2^{2^{7|\varphi|}}$ and period $\widetilde{y} < 2|\varphi| \cdot 2^{2^{7|\varphi|}} \cdot 2^{2^{7|\varphi|}}$.*

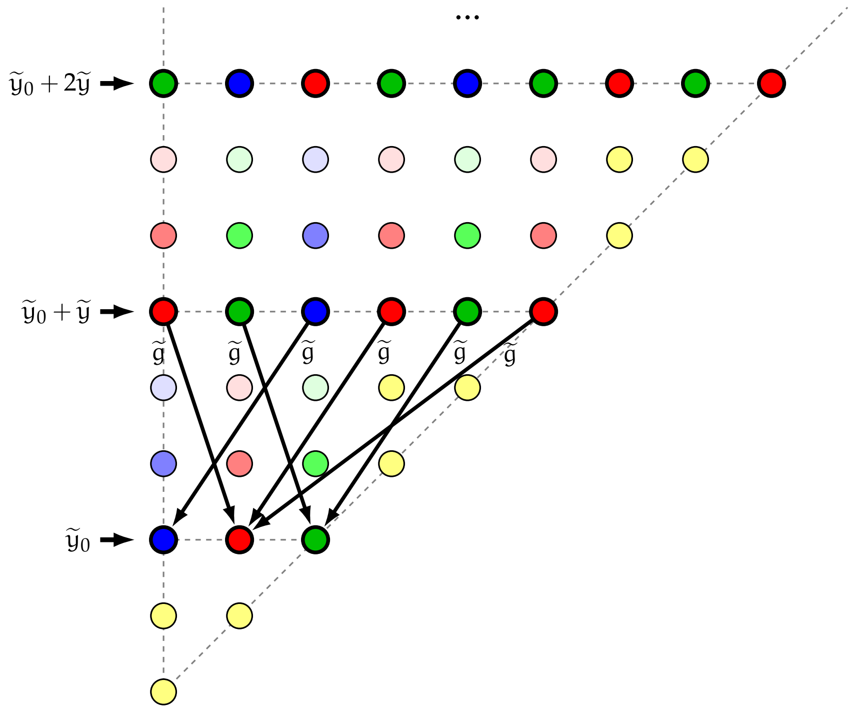


Figure 3: A periodic compass structure with threshold $\tilde{y}_0$, period $\tilde{y}$, and binding $\tilde{g}$.

4. Tight complexity bounds to the satisfiability problem for $\mathsf{AB\bar{B}}$

In this section, we show that the satisfiability problem for $\mathsf{AB\bar{B}}$ interpreted over (either finite or infinite) interval temporal structures is EXPSPACE-complete.

The EXPSPACE-hardness of the satisfiability problem for $\mathsf{AB\bar{B}}$ follows from a reduction from the *exponential-corridor tiling problem*, which is known to be EXPSPACE-complete [19]. Formally, an instance of the exponential-corridor tiling problem is a tuple $\mathcal{T} = (T, t_\perp, t_\top, H, V, n)$ consisting of a finite set T of tiles, a bottom tile $t_\perp \in T$, a top tile $t_\top \in T$, two binary relations H, V over T (specifying the horizontal and vertical constraints), and a positive natural number n (represented in unary notation). The problem consists in deciding whether there exists a tiling $f : \mathbb{N} \times \{0, ..., 2^n - 1\} \rightarrow T$ of the infinite discrete corridor of height 2^n, that associates the tile $t_\perp$ (resp., $t_\top$) with the bottom (resp., top) row of the corridor and that respects the horizontal and vertical constraints H and V, namely,

i) for every $x \in \mathbb{N}$, we have $f(x, 0) = t_\perp$,

ii) for every $x \in \mathbb{N}$, we have $f(x, 2^n - 1) = t_\top$,

iii) for every $x \in \mathbb{N}$ and every $0 \leqslant y < 2^n$, we have $f(x, y)\ H\ f(x + 1, y)$,

iv) for every $x \in \mathbb{N}$ and every $0 \leqslant y < 2^n - 1$, we have $f(x, y)\ V\ f(x, y + 1)$.

The proof of the following lemma, which reduces the exponential-corridor tiling problem to the satisfiability problem for $\mathsf{AB\bar{B}}$, can be found in [17]. Intuitively, such a reduction exploits (i) the correspondence between the points $p = (x, y)$ inside the infinite corridor $\mathbb{N} \times \{0, ..., 2^n - 1\}$ and the intervals of the form $I_p = [y + 2^n x, y + 2^n x + 1]$, (ii) $|T|$ propositional variables which represent the tiling function f, (iii) n additional propositional variables which represent (the binary expansion of) the y-coordinate of each row of the corridor, and

(iv) the modal operators $\langle \mathsf{A} \rangle$ and $\langle \mathsf{B} \rangle$ by means of which one can enforce the local constrains over the tiling function f (as a matter of fact, this shows that the satisfiability problem for the AB fragment is already hard for EXPSPACE).

Lemma 4.1. *There is a polynomial-time reduction from the exponential-corridor tiling problem to the satisfiability problem for* $\mathsf{AB\bar{B}}$.

As for the EXPSPACE-completeness, we claim that the existence of a compass structure $\mathcal{G}$ that features a given formula φ can be decided by verifying suitable local (and stronger) consistency conditions over all pairs of contiguous rows. In fact, in order to check that these local conditions hold between two contiguous rows y and $y+1$, it is sufficient to store into memory a bounded amount of information, namely, (i) a counter y that ranges over $\{1, ..., 2^{2^{7|\varphi|}} + |\varphi| \cdot 2^{2^{7|\varphi|}}\}$, (ii) the two guessed shadings $\mathcal{S}$ and $\mathcal{S}'$ associated with the rows y and $y+1$, and (iii) a function $g : \mathcal{S} \rightarrow \mathcal{S}'$ that captures the horizontal alignment relation between points with an associated atom from $\mathcal{S}$ and points with an associated atom from $\mathcal{S}'$. This shows that the satisfiability problem for $\mathsf{AB\bar{B}}$ can be decided in exponential space, as claimed by the following lemma. Further details about the decision procedure, including soundness and completeness proofs, can be found in [17].

Lemma 4.2. *There is an EXPSPACE non-deterministic procedure that decides whether a given formula of* $\mathsf{AB\bar{B}}$ *is satisfiable or not.*

Summing up, we obtain the following tight complexity result.

Theorem 4.3. *The satisfiability problem for* $\mathsf{AB\bar{B}}$ *interpreted over (prefixes of) natural numbers is EXPSPACE-complete.*

5. Conclusions

In this paper, we proved that the satisfiability problem for $\mathsf{AB\bar{B}}$ interpreted over (prefixes of) the natural numbers is EXPSPACE-complete. We restricted our attention to these domains because it is a common commitment in computer science. Moreover, this gave us the possibility of expressing meaningful metric constraints in a fairly natural way. Nevertheless, we believe it possible to extend our results to the class of all linear orderings as well as to relevant subclasses of it. Another restriction that can be relaxed is the one about singleton intervals: all results in the paper can be easily generalized to include singleton intervals in the underlying structure $\mathbb{I}_{\mathbb{N}}$. The most exciting challenge is to establish whether the modality $\bar{\mathsf{A}}$ can be added to $\mathsf{AB\bar{B}}$ preserving decidability (and complexity). It is easy to show that there is not a straightforward way to lift the proof for $\mathsf{AB\bar{B}}$ to $\mathsf{AB\bar{B}\bar{A}}$ (notice that $\langle \mathsf{A} \rangle$, $\langle \mathsf{B} \rangle$, and $\langle \bar{\mathsf{B}} \rangle$ are all future modalities, while $\langle \bar{\mathsf{A}} \rangle$ is a past one).

References

[1] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the Association for Computing Machinery*, 26(11):832–843, 1983.

[2] D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. Decidable and undecidable fragments of Halpern and Shoham's interval temporal logic: towards a complete classification. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 5330 of *Lecture Notes in Computer Science*, pages 590–604. Springer, 2008.

[3] D. Bresolin, V. Goranko, A. Montanari, and P. Sala. Tableau-based decision procedures for the logics of subinterval structures over dense orderings. *Journal of Logic and Computation*, doi:10.1093/logcom/exn063, 2008.

[4] D. Bresolin, V. Goranko, A. Montanari, and G. Sciavicco. On decidability and expressiveness of propositional interval neighborhood logics. In *Proceedings of the International Symposium on Logical Foundations of Computer Science (LFCS)*, volume 4514 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2007.

[5] D. Bresolin, V. Goranko, A. Montanari, and G. Sciavicco. Propositional interval neighborhood logics: expressiveness, decidability, and undecidable extensions. *Annals of Pure and Applied Logic*, 161(3):289–304, 2009.

[6] D. Bresolin, A. Montanari, P. Sala, and G. Sciavicco. Optimal tableaux for right propositional neighborhood logic over linear orders. In *Proceedings of the 11th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 5293 of *Lecture Notes in Artificial Intelligence*, pages 62–75. Springer, 2008.

[7] D. Bresolin, A. Montanari, and G. Sciavicco. An optimal decision procedure for Right Propositional Neighborhood Logic. *Journal of Automated Reasoning*, 38(1-3):173–199, 2007.

[8] D. Bresolin, V. Goranko A. Montanari, and G. Sciavicco. Right propositional neighborhood logic over natural numbers with integer constraints for interval lengths. In *Proceedings of the 7th IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, pages 240–249. IEEE Comp. Society Press, 2009.

[9] D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: mathematical foundations and computational aspects*. Oxford University Press, 1994.

[10] V. Goranko, A. Montanari, and G. Sciavicco. Propositional interval neighborhood temporal logics. *Journal of Universal Computer Science*, 9(9):1137–1167, 2003.

[11] V. Goranko, A. Montanari, and G. Sciavicco. A road map of interval temporal logics and duration calculi. *Applied Non-classical Logics*, 14(1-2):9–54, 2004.

[12] J.Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the Association for Computing Machinery*, 38:279–292, 1991.

[13] I. Hodkinson, A. Montanari, and G. Sciavicco. Non-finite axiomatizability and undecidability of interval temporal logics with C, D, and T. In *Proceedings of the 17th Annual Conference of the EACSL*, volume 5213 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 2008.

[14] K. Lodaya. Sharpening the undecidability of interval temporal logic. In *Proceedings of the 6th Asian Computing Science Conference on Advances in Computing Science*, volume 1961 of *Lecture Notes in Computer Science*, pages 290–298. Springer, 2000.

[15] C. Lutz and F. Wolter. Modal logics of topological relations. *Logical Methods in Computer Science*, 2(2), 2006.

[16] A. Montanari, G. Puppis, and P. Sala. A decidable spatial logic with cone-shaped cardinal directions. In *Proceedings of the 18th Annual Conference of the EACSL*, volume 5771 of *Lecture Notes in Computer Science*, pages 394–408. Springer, 2009.

[17] A. Montanari, G. Puppis, P. Sala, and G. Sciavicco. Decidability of the interval temporal logic $\mathsf{AB\bar{B}}$ over the natural numbers. Research Report UDMI/2009/07, Department of Mathematics and Computer Science, University of Udine, Udine, Italy, 2009, http://users.dimi.uniud.it/~angelo.montanari/rr200907.pdf.

[18] M. Otto. Two variable first-order logic over ordered domains. *Journal of Symbolic Logic*, 66(2):685–702, 2001.

[19] P. Van Emde Boas. The convenience of tilings. In *Complexity, Logic and Recursion Theory*, volume 187 of *Lecture Notes in Pure and Applied Mathematics*, pages 331–363. Marcel Dekker Inc., 1997.

[20] Y. Venema. A modal logic for chopping intervals. *Journal of Logic and Computation*, 1(4):453–476, 1991.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 609-620
www.stacs-conf.org

RELAXED SPANNERS FOR DIRECTED DISK GRAPHS

DAVID PELEG [1] AND LIAM RODITTY [2]

[1] Department of Computer Science and Applied Mathematics,
The Weizmann Institute of Science, Rehovot 76100, Israel
E-mail address: david.peleg@weizmann.ac.il

[2] Department of Computer Science, Bar-Ilan University,
Ramat-Gan 52900, Israel
E-mail address: liamr@macs.biu.ac.il

Abstract. Let (V, δ) be a finite metric space, where V is a set of n points and δ is a distance function defined for these points. Assume that (V, δ) has a constant doubling dimension d and assume that each point $p \in V$ has a disk of radius $r(p)$ around it. The disk graph that corresponds to V and $r(\cdot)$ is a *directed* graph $I(V, E, r)$, whose vertices are the points of V and whose edge set includes a directed edge from p to q if $\delta(p, q) \leq r(p)$. In [8] we presented an algorithm for constructing a $(1 + \epsilon)$-spanner of size $O(n/\epsilon^d \log M)$, where M is the maximal radius $r(p)$. The current paper presents two results. The first shows that the spanner of [8] is essentially optimal, i.e., for metrics of constant doubling dimension it is not possible to guarantee a spanner whose size is independent of M. The second result shows that by slightly relaxing the requirements and allowing a small perturbation of the radius assignment, considerably better spanners can be constructed. In particular, we show that if it is allowed to use edges of the disk graph $I(V, E, r_{1+\epsilon})$, where $r_{1+\epsilon}(p) = (1+\epsilon)\cdot r(p)$ for every $p \in V$, then it is possible to get a $(1 + \epsilon)$-spanner of size $O(n/\epsilon^d)$ for $I(V, E, r)$. Our algorithm is simple and can be implemented efficiently.

Introduction

This paper concerns efficient constructions of spanners for disk graphs, an important family of directed graphs. A *spanner* is essentially a skeleton of the graph, namely, a sparse spanning subgraph that faithfully represents distances. Formally, a subgraph H of a graph G is a t-spanner of G if $\delta_H(u, v) \leq t \cdot \delta_G(u, v)$ for every two nodes u and v, where $\delta_{G'}(u, v)$ denotes the distance between u and v in G'. We refer to t as the *stretch factor* of the spanner. Graph spanners have received considerable attention over the last two decades, and were used implicitly or explicitly as key ingredients of various distributed applications. It is known how to efficiently construct a $(2k - 1)$-spanner of size $O(n^{1+1/k})$ for every weighted undirected graph, and this size-stretch tradeoff is conjectured to be tight. Baswana and

1998 ACM Subject Classification: F.2 ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY, F.2.0 General .

Key words and phrases: Spanners, Directed graphs.

Thanks: Supported in part by grants from the Minerva Foundation and the Israel Ministry of Science.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2489

Sen [**?**] presented a linear time randomized algorithm for computing such a spanner. In directed graphs, however, the situation is different. No such general size-stretch tradeoff can exist, as indicated by considering the example of a directed bipartite graph G in which all the edges are directed from one side to the other; clearly, the only spanner of G is G itself, as any spanner for G must contain every edge.

The main difference between undirected and directed graphs is that in undirected graphs the distances are symmetric, that is, a path of a certain length from u to v can be used also from v to u. In directed graphs, however, the existence of a path from u to v does not imply anything on the distance in the opposite direction from v to u. Hence, in order to obtain a spanner for a directed graph one must impose some restriction either on the graph or on its distances. In order to bypass the problem of asymmetric distances of directed graphs, Cowen and Wagner [5] introduced the notion of *roundtrip distances* in which the distance between u and v is composed of the shortest path from u to v plus the shortest path from v to u. It is easy to see that under this definition distances are symmetric also in directed graphs. It is shown by Cowen and Wagner [5] and later by Roditty, Thorup and Zwick [6] that methods of path approximations from undirected graphs can work using more ideas also in directed graphs when roundtrip distances are considered. Bollobás, Coppersmith and Elkin [**?**] introduced the notion of *distance preservers* and showed that they exist also in directed graphs.

In [8] we presented a spanner construction for directed graphs without symmetric distances. The restriction that we imposed on the graph was that it must be a disk graph. More formally, let (V, δ) be a finite metric space of constant doubling dimension d, where V is a set of n points and δ is a distance function defined for these points. A metric is said to be of *constant doubling dimension* if a ball with radius r can be covered by at most a constant number of balls of radius $r/2$. Every point $p \in V$ is assigned with a radius $r(p)$. The disk graph that corresponds to V and $r(\cdot)$ is a directed graph $I(V, E, r)$, whose vertices are the points of V and whose edge set includes a directed edge from p to q if q is inside the disk of p, that is, $\delta(p, q) \leq r(p)$. In [8] we presented an algorithm for constructing a $(1 + \epsilon)$-spanner with size $O(n/\epsilon^d \log M)$, where M is the maximal radius. In the case that we remove the radius restriction the resulted graph is the complete undirected graph where the weight of every edge is the distance between its endpoint. In such a case it is possible to create $(1 + \epsilon)$-spanners of size $O(n/\epsilon^d)$, see [4], [2] and [9] for more details. Moreover, when the radii are all the same and the graph is the unit disk graph then it is also possible to create $(1 + \epsilon)$-spanners of size $O(n/\epsilon^d)$, see [3], [8].

As a result of that, a natural question is whether a spanner size of $O(n/\epsilon^d \log M)$ in the case of directed disk graph is indeed the best possible or maybe it is possible to get a spanner of size $O(n/\epsilon^d)$ as in the cases of the complete graph and the unit disk graph. For the case of the Euclidean metric space, the answer turns out to be positive; a simple modification of the Yao graph construction [11] to fit the directed case yields a directed spanner of size $O(n/\epsilon^d)$. However, the question remains for more general metric spaces, and in particular for the important family of metric spaces of bounded doubling dimension.

In this paper we provide an answer for this question. We show that our construction from [8] is essentially optimal by providing a metric space with a constant doubling dimension and a radius assignment whose corresponding disk graph has $\Omega(n^2)$ edges and none of its edges can be removed. (This does not contradict our spanner construction from [8] as the maximal radius in that case is $\Theta(2^n)$ and hence $\log M = n$.)

This (essentially negative) optimality result motivates our main interest in the current paper, which focuses on attempts to slightly relax the assumptions of the model, in order to obtain sparser spanner constructions. Indeed, it turns out that such sparser spanner constructions are feasible under a suitably relaxed model. Specifically, we demonstrate the fact that if a small perturbation of the radius assignment is allowed, then a $(1+\epsilon)$-spanner of size $O(n/\epsilon^d)$ is attainable. More formally, we show that if we are allowed to use edges of the disk graph $I(V, E, r_{1+\epsilon})$, where $r_{1+\epsilon}(p) = (1+\epsilon) \cdot r(p)$ for every $p \in V$, then it is possible to get a $(1+\epsilon)$-spanner of size $O(n/\epsilon^d)$ for the original disk graph $I(V, E, r)$. This approach is similar in its nature to the notation of *emulators* introduced by Dor, Halperin and Zwick [1]. An emulator of a graph may use any edge that does not exist in the graph in order to approximate its distances. It was used in the context of spanners with an additive stretch.

The main application of disk graph spanners is for topology control in the wireless ad hoc network model. In this model the power required for transmitting from p to q is commonly taken to be $\delta(p, q)^\alpha$, where $\delta(p, q)$ denotes the distance between p and q and α is a constant typically assumed to be between 2 and 4. Most of the ad hoc network literature makes the assumption that the transmission range of all nodes is identical, and consequently represents the network by a *unit disk graph* (UDG), namely, a graph in which two nodes p, q are adjacent if their distance satisfies $\delta(p, q) \leq 1$. A unit disk graph can have as many as $O(n^2)$ edges.

There is an extensive body of literature on spanners of unit disk graphs. Gao et al. [3], Wang and Yang-Li [10] and Yang-Li et al. [7] considered the restricted Delaunay graph, whose worst-case stretch is constant (larger than $1+\epsilon$). In [8] we showed that any $(1+\epsilon)$-geometric spanner can be turned into a $(1+\epsilon)$-UDG spanner.

Disk graphs are a natural generalization of unit disk graphs, that provide an intermediate model between the complete graph and the unit disk graph. Our size efficient spanner construction for disk graphs whose radii are allowed to be slightly larger falls exactly into the model of networks in which the stations can change their transmission power. In particular our constriction implies that if any station increases its transmission power by a small fraction then a considerably improved topology can be built for the network.

Our result has both practical and theoretical implications. From a practical point of view it shows that, in certain scenarios, extending the transmission radii even by a small factor can significantly improve the overall quality of the network topology. The result is also very intriguing from a theoretical standpoint, as to the best of our knowledge, our relaxed spanner is the first example of a spanner construction for directed graphs that enjoys the same properties as the best constructions for undirected graphs. (As mentioned above, it is easy to see that for general directed graphs, it is not possible to have an algorithm that given any directed graph produces a sparse spanner for it.) In that sense, our result can be viewed as a significant step towards gaining a better understanding for some of the fundamental differences between directed and undirected graphs. Our result also opens several new research directions in the relaxed model of disk graphs. The most obvious research questions that arise are whether it is possible to obtain other objects that are known to exist in undirected graphs, such as compact routing schemes and distance oracles, for disk graphs as well.

The rest of this paper is organized as follows. In the next section we present a metric space of constant doubling dimension in which no edge can be removed from its corresponding disk graph. Section 2 first describes a simple variant of our construction from [8],

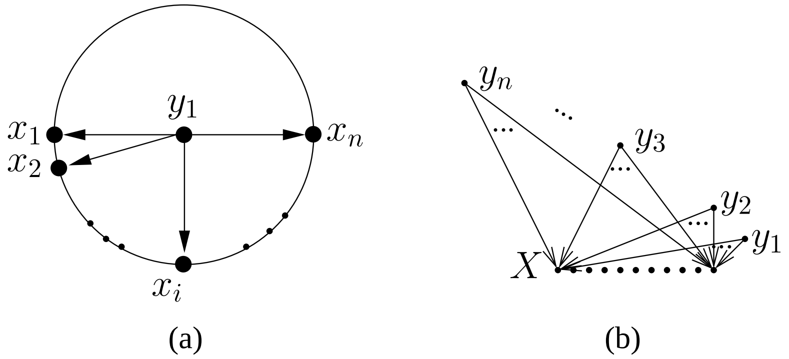


Figure 1: (a) First step in constructing the non-sparsifiable disk graph G. (b) The non-sparsifiable disk graph G.

and then uses it together with new ideas in order to obtain our new relaxed construction. Finally, in Section 3 we present some concluding remarks and open problems.

1. Optimality of the spanner construction

In this section we build a disk graph G with $2n$ vertices and $\Omega(n^2)$ edges that is non-sparsifiable, namely, whose only spanner is G itself. In this graph $M = \Omega(2^n)$ hence our spanner construction from [8] has a size of $\Omega(n^2)$ and is essentially optimal.

Given a set of points, we present a distance function such that for a given assignment of radii for the points any spanner of the resulting disk graph must have $\Omega(n^2)$ edges. We then prove that the underlying metric space has a constant doubling dimension.

We partition the points into two types, $Y = \{y_1, \ldots, y_n\}$ and $X = \{x_1, \ldots, x_n\}$. We now define the distance function $\delta(\cdot, \cdot)$ and the radii assignment $r(\cdot)$. The main idea is to create a bipartite graph $G(X, Y, E)$ in which every point of Y is connected by a directed edge to all the points of X.

The distance between any two points x_i and x_j is at least $1+\epsilon$ for some small $0 < \epsilon < 1$ and the radius assignment of every point x_i is exactly 1. Thus, there are no edges between the points of X.

We now define the distances between the points of Y and the points of X. We start with the point y_1. Let $\delta(y_1, x_i) = n$ for every $x_i \in X$ and let $r(y_1) = n$. Place the points of X on the boundary of a ball of radius n centered at y_1 such that the distance between any two consecutive points x_i and x_{i+1} is exactly $1 + \epsilon$. This is depicted in Figure 1(a).

Turning to the point y_2, let $\delta(y_2, x_i) = 2n$ for every $x_i \in X$, $\delta(y_2, y_1) = 2n + \epsilon$, and $r(y_2) = 2n$. Hence there is an edge from y_2 to all the points of X, but no edge connects y_2 and y_1.

We now turn to define the general case. Consider $y_i \in Y$. Let $r(y_i) = 2^{i-1}n$ and $\delta(y_i, x_j) = 2^{i-1}n$ for every $x_j \in X$. Let $\delta(y_i, y_{i-1}) = 2^{i-1}n + \epsilon$, and in general, for every $0 < j < i$ we have

$$\delta(y_i, y_j) \; = \; \sum_{k=j}^{i-1} \delta(y_{k+1}, y_k) \; , \tag{1.1}$$

implying that

$$\delta(y_i, y_j) \; < \; 2^i n. \tag{1.2}$$

It is easy to verify that y_i has outgoing edges to the points of X (and to them only) and it does not have any incoming edges. See Figure 1(b).

The resulting disk graph G has $2n$ vertices and $\Omega(n^2)$ edges. Clearly, removing any edge from G will increase the distance between its head and its tail to infinity, and thus the only spanner of G is G itself.

It is left to show that the metric space defined above for G has a constant doubling dimension. Given a metric space (V, δ), its *doubling dimension* is defined to be the minimal value d such that every ball B of radius r in the metric space can be covered by 2^d balls of radius $r/2$. In the next Theorem we prove that for the metric space described above, d is constant.

Theorem 1.1. *The metric space $(X \cup Y, \delta)$ defined for G has a constant doubling dimension.*

Proof. Let B be a ball with an arbitrary radius r. We show that it is possible to cover all the points of $X \cup Y$ within B using a constant number of balls whose radius is $r/2$. The proof is divided into two cases.

Case a: There is some $y_j \in Y$ within the ball B. (If there is more than one such point, then let y_j be the point whose index is maximal.) Let B' be a ball of radius $R = 2r$ centered at y_j. Clearly $B \subset B'$, so B' contains all the points of B. In what follows we show that all the points of $X \cup Y$ within B' can be covered by a constant number of balls of radius $r/2$. Let y_i be the point within B' whose index is maximal. We have to consider two possible scenarios. The first is that $y_j = y_i$. This implies that $y_{j+1} \notin B'$, hence $R < \delta(y_{j+1}, y_j) = 2^j n + \epsilon$. We now show that it is possible to cover B' by a constant number of balls of radius $R/4$. If $R < 2^{j-1}n$, then only y_j is within B' and it is covered by a ball of radius $R/4$ centered at itself. If $2^{j-1}n \leq R < 2^{j-1}n + \epsilon$, then B' contains all the points of X and y_j. From packing arguments it follows that it is possible to cover all the points of X by a constant number of balls of radius $n/4$, hence also by a constant number of balls of radius $R \geq n$. The point y_j itself is covered by a ball centered at it. Finally, if $2^{j-1}n + \epsilon \leq R < 2^j n + \epsilon$, then $R/4$ is at least $2^{j-3}n + \epsilon/4$. A ball centered at y_{j-3} of radius $R/4$ covers every y_k within B', where $1 \leq k \leq j-3$, as $\delta(y_{j-3}, y_k) \leq 2^{j-3}n$. Hence, we cover $Y \cap B'$ by balls of radius $R/4$ whose centers are y_j, y_{j-1}, y_{j-2} and y_{j-3}. We cover $X \cap B'$ as before. This completes the first scenario, where $y_i = y_j$. Assume now that $y_i \neq y_j$. This implies that $\delta(y_i, y_j) \leq R$ and that $R < \delta(y_{i+1}, y_j)$, where the first inequality follows from the fact that $y_i \in B'$ and the second inequality follows from the fact that y_i is the point with maximal index inside B', hence, $y_{i+1} \notin B'$. As $\delta(y_i, y_{i-1}) \leq \delta(y_i, y_j)$, we get that $2^{i-1}n + \epsilon \leq R$. Also, by (1.2), $\delta(y_{i+1}, y_j) < 2^{i+1}n$. We conclude that $2^{i-1}n \leq R < 2^{i+1}n$ and that $R/4 \geq 2^{i-3}n$. A ball centered at y_{i-3} of radius $R/4$ covers every y_k within B', where $k \leq i-3$, as $\delta(y_{i-3}, y_k) \leq 2^{i-3}n$. Hence, we can cover $B' \cap Y$ by balls of radius $R/4$ whose centers are y_i, y_{i-1}, y_{i-2} and y_{i-3}. We cover $X \cap B'$ as before. This completes the first case.

Case b: The ball B does not contain any point from Y. The points of X are spread as appears in Figure 1(a), thus by standard packing arguments, any ball that contains only points from X is covered by a constant number of balls of half the radius. ■

2. Improved spanner in the relaxed disk graph model

The (negative) optimality result from the previous section motivates us to look for a slightly relaxed definition of disk graphs in which it will still be possible to create a spanner of size $O(n/\epsilon^d)$.

Let (V, δ) be a metric space of constant doubling dimension d with a radius assignment $r(\cdot)$ for its points and let $I = (V, E, r)$ be its corresponding disk graph. Assume that we multiply the radius assignment of every point by a factor of $1 + \epsilon$, for some $\epsilon > 0$, and let $I' = (V, E', r_{1+\epsilon})$ be the corresponding disk graph. It is easy to see that $E \subseteq E'$. In this section we show that it is possible to create a $(1 + \epsilon)$-spanner of size $O(n/\epsilon^d)$ if we are allowed to use edges of I'. As a first step we present a simple variant of our $(1 + \epsilon)$-spanner construction of size $O(n/\epsilon^d \log M)$ from [8]. This variation is needed in order to obtain the efficient construction in the relaxed model which is presented right afterwards.

2.1. Spanners for general disk graphs

Let (V, δ) be a metric space of constant doubling dimension and assume that any point $p \in V$ is the center of a ball of radius $r(p)$, where $r(p)$ is taken from the range $[1, M]$. In this section we describe a simple variant of our construction from [8], which computes a $(1 + \epsilon)$-spanner with $O(n/\epsilon^d \log M)$ edges for a given disk graph. We then use this variant, together with new ideas, in order to obtain (in the next section) our main result, namely, a spanner with only $O(n/\epsilon^d)$ edges.

The spanner construction algorithm receives as input a directed graph $I(V, E, r)$ and an arbitrarily small (constant) approximation factor $\epsilon > 0$, and constructs a set of spanner edges $E_{\mathrm{SP}}^{\mathrm{DIR}}$, returning the spanner subgraph $H^{\mathrm{DIR}}(V, E_{\mathrm{SP}}^{\mathrm{DIR}})$. The construction of the spanner is based on a hierarchical partition of the points of V that takes into account the different radius of each point. The construction operates as follows. Let α and β be two small constants depending on ϵ, to be fixed later on. Assume that the ball radii are scaled so that the smallest edge in the disk graph is of weight 1. Let i be an integer from the range $[0, \lfloor \log_{1+\alpha} M \rfloor]$ and let $M_i = M/(1 + \alpha)^i$. The edges of $I(V, E, r)$ are partitioned into classes by length, letting $E(M_{i+1}, M_i) = \{(x, y) \mid M_{i+1} \leq \delta(x, y) \leq M_i\}$. Let $\ell(x, y)$ be the level of the edge (x, y), that is, $\ell(x, y) = i$ such that $(x, y) \in E(M_{i+1}, M_i)$. Let p be a point whose ball is of radius $r(p) \in [M_{i+1}, M_i]$. It follows that level i is the first level in which p can have outgoing edges. We denote this level by $\ell(p)$.

For every $i \in [0, \lfloor \log_{1+\alpha} M \rfloor]$, starting from $i = 0$, the edges of the class $E(M_{i+1}, M_i)$ are considered by the algorithm in a non-decreasing order. (Assume that in each class the edges are sorted by their weight.) In each stage of the construction we maintain a set of pivots P_i. Let $x \in V$ and let $\mathsf{NN}(x, P_i)$ be the nearest neighbor of x among the points of P_i. For a pivot $p \in P_i$, define $\Gamma_i(p) = \{x \mid x \in V, \mathsf{NN}(x, P_i) = p, r(x) \geq \delta(x, p)\}$, namely, all the points that have a directed edge to p and p is their nearest neighbor from P_i. We refer to $\Gamma_i(p)$ as the *close neighborhood* of p.

The algorithm is given in Figure 2. Let (x, y) be an edge considered by the algorithm in the ith iteration. The algorithm first checks whether x or y or both should be added to the pivots set P_i. The main change with respect to [8] is that if y is assigned with a large enough radius it might become a pivot when the edge (x, y) is examined. When considering the edge (x, y), the algorithm acts according to the following rule: If the distance from x to its nearest neighbor in P_i is greater than βM_{i+1} then x is added to P_i. If the distance from y to its nearest neighbor in P_i is greater than βM_{i+1} and the radius of y

```
Algorithm disk-spanner $(I(V,E,R),\epsilon)$
$E_{\mathrm{SP}}^{\mathrm{DIR}} \leftarrow \phi$
$P_0 \leftarrow \phi$
for  $i \leftarrow 0$ to $\lfloor \log_{1+\alpha} M \rfloor$
      for  each $(x,y) \in E(M_{i+1}, M_i)$ do
            if $\delta(\mathsf{NN}(x,P_i),x) > \beta M_{i+1}$ then $P_i \leftarrow P_i \cup \{x\}$
            if $\delta(\mathsf{NN}(y,P_i),y) > \beta M_{i+1} \wedge r(y) \geq M_{i+1}$ then $P_i \leftarrow P_i \cup \{y\}$
            if $r(y) \geq M_{i+1}$
              if $\nexists (x',y') \in E_{\mathrm{SP}}^{\mathrm{DIR}}$ s.t. $x' \in \Gamma_i(\mathsf{NN}(x,P_i)) \wedge y' \in \Gamma_i(\mathsf{NN}(y,P_i))$
                then $E_{\mathrm{SP}}^{\mathrm{DIR}} \leftarrow E_{\mathrm{SP}}^{\mathrm{DIR}} \cup \{(x,y)\}$
            if $r(y) < M_{i+1}$
              if $\nexists (x',y) \in E_{\mathrm{SP}}^{\mathrm{DIR}}$ s.t. $x' \in \Gamma_i(\mathsf{NN}(x,P_i))$
                then $E_{\mathrm{SP}}^{\mathrm{DIR}} \leftarrow E_{\mathrm{SP}}^{\mathrm{DIR}} \cup \{(x,y)\}$
      $P_{i+1} \leftarrow P_i$
return $H^{\mathrm{DIR}}(V, E_{\mathrm{SP}}^{\mathrm{DIR}})$
```

Figure 2: A high level implementation of the spanner construction algorithm for *general* disk graphs

is at least M_{i+1} then y is added to P_i. To decide whether the edge (x,y) is added to the spanner, the following two cases are considered. The first case is when $r(y) \geq M_{i+1}$. In this case, if there is no edge from the close neighborhood of x to the close neighborhood of y then (x,y) is added to the spanner. The second case is when $r(y) < M_{i+1}$. In this case, if there is no edge from the close neighborhood of x to y then (x,y) is added to the spanner. When i reaches $\lfloor \log_{1+\alpha} M \rfloor$, the algorithm handles all the edges that belong to $E(M_{\lfloor \log_{1+\alpha} M \rfloor + 1}, M_{\lfloor \log_{1+\alpha} M \rfloor})$. This includes also edges whose weight is 1, the minimal possible weight. The algorithm returns the directed graph $H^{\mathrm{DIR}}(V, E_{\mathrm{SP}}^{\mathrm{DIR}})$.

In what follows we prove that for suitably chosen α and β, $H^{\mathrm{DIR}}(V, E_{\mathrm{SP}}^{\mathrm{DIR}})$ is a $(1+\epsilon)$-spanner with $O(n/\epsilon^d \log M)$ edges of the directed graph $I(V,E,r)$.

Lemma 2.1 (Stretch)**.** *Let $\epsilon > 0$, set $\alpha = \beta < \epsilon/6$ and let $H = H^{DIR}(V, E_{SP}^{DIR})$ be the graph returned by Algorithm* **disk-spanner**$(I(V,E,r),\epsilon)$. *If $(x,y) \in E$ then $\delta_H(x,y) \leq (1+\epsilon)\delta(x,y)$.*

Proof. Recall that the radii are scaled so that the shortest edge is of weight 1. We prove that every directed edge of an arbitrary node $x \in V$ is approximated with $1+\epsilon$ stretch. Let $i \in [0, \lfloor \log_{1+\alpha} M \rfloor]$. The proof is by induction on i. For a given node x, the base of the induction is the maximal value of i in which x has an edge in $E(M_{i+1}, M_i)$. Let j be this value for x, that is, the set $E(M_{j+1}, M_j)$ contains the shortest edge that touches x. Every other node is at distance at least M_{j+1} away from x, hence x is a pivot at this stage and every edge that touches x from the set $E(M_{j+1}, M_j)$ is added to $E_{\mathrm{SP}}^{\mathrm{DIR}}$.

Let $(x,y) \in E(M_{i+1}, M_i)$ for some $i < j$ and let $p = \mathsf{NN}(x, P_i)$. Assume that $r(y) \geq M_{i+1}$ and let $q = \mathsf{NN}(y, P_i)$. It follows from definition that $\delta(x,p) \leq \beta M_{i+1}$ and $\delta(y,q) \leq \beta M_{i+1}$.

If the edge (x,y) is not in the spanner, then there must be an edge $(\hat{x}, \hat{y}) \in E_{\mathrm{SP}}^{\mathrm{DIR}}$, where $\hat{x} \in \Gamma_i(p)$ and $\hat{y} \in \Gamma_i(q)$. The crucial observation is that the radius of x and $\hat{y}$ is at least M_{i+1}. By the choice of β, it follows that $2\beta M_{i+1} < M_{i+1}$ and $(x,\hat{x}), (\hat{y}, y) \in E$. Thus, there is a (directed) path from x to y of the form $\langle x, \hat{x}, \hat{y}, y\rangle$ whose length is $4\beta M_{i+1} + M_i$. However, only its middle edge, $(\hat{x}, \hat{y})$, is in $E_{\text{SP}}^{\text{DIR}}$. The length of this edge is bounded by the length of the edge (x, y) since the algorithm picked the minimal edge that connects between the neighborhoods. This implies that the length of $(\hat{x}, \hat{y})$ is at most M_i.

By the inductive hypothesis, the edges $(x, \hat{x})$ and $(\hat{y}, y)$ whose weight is at most $2\beta M_{i+1}$ are approximated with $1+\epsilon$ stretch. Thus, there is a path in the spanner from x to y whose length is at most $(1+\epsilon)\delta(x, \hat{x}) + M_i + (1+\epsilon)\delta(\hat{y}, y)$, and this can be bounded by

$$(1+\epsilon)4\beta M_{i+1} + M_i \;=\; ((1+\epsilon)4\beta + (1+\alpha))M_{i+1}.$$

As the edge $(x, y) \in E(M_{i+1}, M_i)$ it follows that $\delta(x, y) \geq M_{i+1}$. It remains to prove that $1 + 4\epsilon\beta + 4\beta + \alpha \leq 1 + \epsilon$, which follows directly from the choice of α and β.

If $r(y) < M_{i+1}$ then there must be an edge $(\hat{x}, y) \in E_{\text{SP}}^{\text{DIR}}$, where $\hat{x} \in \Gamma_i(p)$. Following similar arguments to those used above it can be shown that there is a path in the spanner from x to y of length at most $(1+\epsilon)2\beta M_{i+1} + M_i$ and hence bounded by $(1+\epsilon)M_{i+1}$. ∎

The size of the spanner. We now prove that the size of the spanner $H^{\text{DIR}}(V, E_{\text{SP}}^{\text{DIR}})$ is $O(n/\epsilon^d \log M)$. As a first step, we state the following well-known lemma, cf. [2].

Lemma 2.2. *[Packing Lemma] If all points in a set $U \in \mathbb{R}^d$ are at least r apart from each other, then there are at most $(2R/r + 1)^d$ points in U within any ball X of radius R.*

The next lemma establishes a bound on the number of incoming spanner edges that a point may be assigned on stage $i \in [0, \lfloor \log_{1+\alpha} M \rfloor]$ of the algorithm.

Lemma 2.3. *Let $i \in [0, \lfloor \log_{1+\alpha} M \rfloor]$ and let $y \in V$. The total number of incoming edges of y that were added to the spanner on stage i is $O(\epsilon^{-d})$.*

Proof. Let (x, y) be a spanner edge and let $\mathsf{NN}(x, P_i) = p$. We associate (x, y) to p. From the spanner construction algorithm it follows that this is the only incoming edge of y whose source is in $\Gamma_i(p)$. Thus, this is the only incoming edge of y which is associated to p. Now consider all the incoming edges of y on stage i. The source of each of these edges is associated to a unique pivot within distance of at most $M_i + 2\beta M_{i+1}$ away from y and any two pivots are βM_{i+1} apart from each other. Using Lemma 2.2, we get that the number of edges entering y is $(\frac{M_i + 2\beta M_{i+1}}{\beta M_{i+1}} + 1)^d = ((1+\alpha)/\beta + 3)^d = O(\epsilon^{-d})$. ∎

It follows from the above lemma that the total number of edges that were added to $E_{\text{SP}}^{\text{DIR}}$ in the main loop is $O(n/\epsilon^d \log M)$. The total cost of the construction algorithm is $O(m \log n)$. For more details on the construction time see [8].

2.2. Spanner for relaxed disk graphs

Let (V, δ) be a metric space of constant doubling dimension d with a radius assignment $r(\cdot)$ for its points and let $I = (V, E, r)$ be its corresponding disk graph. Assume that we multiply the radius assignment of every point by a factor of $1+\epsilon$, for some $\epsilon > 0$, and let $I' = (V, E', r_{1+\epsilon})$ be the corresponding disk graph. In this section we show that it is possible to create a $(1+\epsilon)$-spanner of I of size $O(n/\epsilon^d)$ if we are allowed to use edges of I'.

Our construction consists of two stages: a building stage and a pruning stage. The building stage creates two spanners, H and H', using the algorithm of Section 2.1, where H is the spanner of I and H' is the spanner of I'. In the pruning stage we prune the union of these two spanners. Throughout the pruning stage we use the radius assignment of each point before the increase. Let $q \in V$ and let $\ell(q)$ be the first level in which q can have outgoing edges, that is, $r(q) \in [M_{\ell(q)+1}, M_{\ell(q)}]$ (recall that as the levels get larger the edges get shorter). In the pruning stage we only prune incoming edges of q whose level is below $\ell(q)$. In other words, we do not touch the incoming edges of q that are shorter than the radius of q. The pruning is done as follow. Let $\gamma = \log_{1+\alpha} 1/\beta + 1$. We keep in the spanner the incoming edges of q that come from the first 4γ different levels below $\ell(q)$.

Let $\hat{H}$ be the resulting spanner and let $\hat{E}$ be the remaining set of edges after the pruning step. In the remainder of this section we show that the size of $\hat{H}$ is $O(n/\epsilon^d)$ and its stretch with respect to the distances in $I(V, E, r)$ is $1+\epsilon$. We start by showing that the size of $\hat{H}$ is $O(n/\epsilon^d)$. Notice that the first part of the proof below is possible only due to the change we have done in the previous section to our spanner construction from [8]. Roughly speaking, given an edge $(p, q) \in E$ that is shorter than $r(q)$ we use pivot selection also on q's side (and not only on p's) to sparisify the graph. This allows us to deal separately with edges of q of length larger than $r(q)$ and those of length smaller than $r(q)$.

Lemma 2.4. $|\hat{E}| = O(n/\epsilon^d)$.

Proof. Let (p, q) be a spanner edge that survived the pruning step. There are two possible cases to consider.

The first case is that $\ell(p, q) > \ell(q)$. Let $i = \ell(p, q)$ and let $x = \mathsf{NN}(p, P_i)$ and $y = \mathsf{NN}(q, P_i)$. By packing considerations similar to Lemma 2.3 it follows that the total number of edges at level i that connects between two pivots as the edge (p, q) that are associated with x (and with y) is $O(1/\epsilon^d)$. The distance between x and y is at most $2\beta M_{i+1} + M_i$, therefore at level $i - 2\gamma$ either x or y are no longer pivots.

Let $x \in P_j$ and $x \notin P_{j-1}$, that is, P_j is the first pivot set that contains x. Then we charge x with every (incoming and outgoing) edge of this type from levels $[j, j + 2\gamma]$ that is incident to x. Now given such an edge (p, q) whose level is i, either x or y are not pivots in level $i - 2\gamma$, which means that either x or y has been charged for this edge, since one of them first becomes a pivot between levels $i - 2\gamma$ and i.

The second case is that $\ell(p, q) \leq \ell(q)$. In this case, it must be that level $\ell(p, q)$ is among the 4γ first different levels below $\ell(q)$ from which an incoming edge is allowed to enter q. Subsequently, we associate the edge (p, q) with q, as the total number of such edges that q can have is $O(\gamma/\epsilon^d)$. ∎

We now turn to prove that the stretch of the spanner $\hat{H}$ with respect to the disk graph I is $1 + \epsilon$.

Lemma 2.5. *Let (p, q) be an edge of the spanner H that was pruned. We show that there is a path in $\hat{H}$ whose length is at most $(1 + \epsilon)\delta(p, q)$.*

Proof. The proof is by induction on the lengths of the pruned edges. For the induction base let (p, q) be the shortest edge that was pruned. For every $x \in V$, let $s(x)$ be the head of an edge whose level is the γ-th level below $\ell(x)$ from which x has an incoming edge. Let $q_1, \ldots q_i, \ldots$ be a sequence of points, where $q_1 = q$ and $q_i = s(q_{i-1})$. As $q_{i+1} = s(q_i)$, it follows that $\ell(q_{i+1}, q_i) \leq \ell(q_i) - \gamma$. Combining this with the fact that $\ell(q_i) \leq \ell(q_i, q_{i-1})$ we get that $\ell(q_{i+1}, q_i) \leq \ell(q_i, q_{i-1}) - \gamma$. Therefore, $\delta(q_i, q_{i-1}) \leq \beta\delta(q_{i+1}, q_i)$.

The analysis distinguishes between two cases.

Case a: There is a point q_t such that $\delta(q_t, q) > \beta\delta(p, q)$. This situation is depicted in Figure 3. (If there is more than one point that satisfies this requirement, take the one whose index is minimal.)

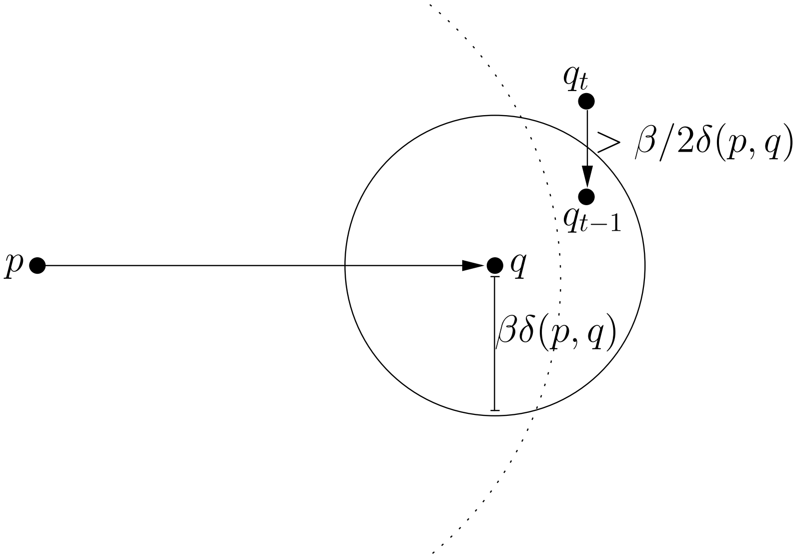


Figure 3: The case in which q_t exists

Claim: $\delta(q_t, q_{t-1}) \geq \frac{\beta}{2}\delta(p,q)$.

Proof. For the sake of contradiction, assume that $\delta(q_t, q_{t-1}) < \frac{\beta}{2}\delta(p,q)$. This implies that

$$2\delta(q_t, q_{t-1}) \ < \ \beta\delta(p,q) \ < \ \delta(q_t, q) \ \leq \ \sum_{i=2}^{t} \delta(q_i, q_{i-1}) \ , \tag{2.1}$$

where the last inequality follows from the triangle inequality as the distance between q and q_t is at most $\sum_{i=2}^{t} \delta(q_{i-1}, q_i)$. For every $2 \leq i \leq t-1$ we have $\delta(q_i, q_{i-1}) \leq \beta\delta(q_{i+1}, q_i)$, which implies that $\delta(q_i, q_{i-1}) \leq \beta^{t-i}\delta(q_t, q_{t-1})$. Combined with (2.1), we get

$$\delta(q_t, q_{t-1}) \ < \ \sum_{i=2}^{t-1} \delta(q_i, q_{i-1}) \ \leq \ \delta(q_t, q_{t-1}) \sum_{i=2}^{t-1} \beta^{t-i} \ .$$

If $\beta < 1/2$ we have $\sum_{i=2}^{t-1} \beta^{t-i} < 1$ and this yields a contradiction. ∎

We now focus our attention on the point q_{t-1}. The minimality of q_t implies that $\delta(q, q_{t-1}) \leq \beta\delta(p,q)$. By combining it with the triangle inequality we get that $\delta(p, q_{t-1}) \leq \delta(p,q) + \beta\delta(p,q)$. Therefore, in the graph I' there must be an edge from p to q_{t-1}.

Let $i = \ell(p, q_{t-1})$. There are two possible scenarios for the spanner H'. The first scenario is when $r'(q_{t-1}) < M_{i+1}$. In this case, there is an edge in H' from some $x \in \Gamma_i(\mathsf{NN}(p,i))$ to q_{t-1}, whose length is at most $\delta(p,q) + \beta\delta(p,q)$.

There are 4γ different levels below $\ell(q_{t-1})$ from which edges that belong to the spanners H and H' are not being pruned and survived to the spanner $\hat{H}$. We know that the edge (q_t, q_{t-1}) is such an edge from the γ-th non-empty level below $\ell(q_{t-1})$. We also know that $\delta(q_t, q_{t-1}) > \frac{\beta}{2}\delta(p,q)$. Therefore, as the length of the edge (x, q_{t-1}) is at most $\delta(p,q) + \beta\delta(p,q)$ it is within the 4γ non-empty levels below $\ell(q_{t-1})$ and it is not pruned. We can now build a path from p to q by concatenating three segments as follows: A path from p to x, the edge (x, q_{t-1}) and a path from q_{t-1} to q. The point x is at most $2\beta\delta(p,q) + 2\beta^2\delta(p,q)$

away from p and for the right choice of β it is less than $\delta(p,q)/(1+\epsilon)$, hence the weight of every edge on the path that approximates the distance between x and p in $H \cup H'$ is less than $\delta(p,q)$, the shortest pruned edge, and the entire path survived the punning stage. Similarly, the point q_{t-1} is at most $\beta\delta(p,q)$ away from q and again for the right choice of β every edge on the path that approximates the distance between q_{t-1} and q survived the punning stage. Thus, we get that there is a path whose length is at most

$$(1+\epsilon)(3\beta\delta(p,q) + 2\beta^2\delta(p,q)) + \delta(p,q) + \beta\delta(p,q) \ ,$$

which is less than $(1+\epsilon)\delta(p,q)$ for $\beta < \epsilon/11$.

The second scenario is when $r'(q_{t-1}) \geq M_{i+1}$. In this case, there is an edge in H' from some $x \in \Gamma_i(\mathsf{NN}(p,i))$ to some $y \in \Gamma_i(\mathsf{NN}(q_{t-1},i))$ whose length is at most $\delta(p,q)+\beta\delta(p,q)$, which is not being pruned. We can build a path from p to q by concatenating three segments as follows: A path from p to x, the edge (x,y) and a path from y to q. As before, for the right choice of β the paths from p to x and from y to q are composed from edges that are shorter from $\delta(p,q)$, the length of the shortest pruned edge, hence, from the minimality $\delta(p,q)$ every edge on these paths survived the punning stage. We get that there is a path whose length is at most

$$(1+\epsilon)(4\beta\delta(p,q) + 5\beta^2\delta(p,q)) + \delta(p,q) + \beta\delta(p,q) \ ,$$

which is less than $(1+\epsilon)\delta(p,q)$ for $\beta < \epsilon/19$. This completes the proof for case a.

Case b: There is no point q_t such that $\delta(q_t,q) > \beta\delta(p,q)$. In this case, let q_{t-1} be the last point in the sequence of points $q_1, \dots q_i, \dots$, where $q_i = s(q_{i-1})$ and $q_1 = q$. Similarly to before, there are two possible scenarios for the spanner H'. Let $i = \ell(p, q_{t-1})$. The first scenario is when $r'(q_{t-1}) < M_{i+1}$. In this case, there is an edge in H' from some $x \in \Gamma_i(\mathsf{NN}(p,i))$ to q_{t-1} whose length is at most $\delta(p,q)+\beta\delta(p,q)$. This edge could not be pruned, since if it was pruned then q_{t-1} could not have been the last point in the sequence. Hence we can construct a path from p to q exactly as we have done in the first scenario of case a, described above. The second scenario is when $r'(q_{t-1}) \geq M_{i+1}$. In this case, we can construct a path from p to q exactly as we have done in the second scenario of case a, described above.

This completes the proof of the induction base. The proof of the general inductive step is almost identical. The only difference is that when a path is constructed from p to q, its portions from p to x and from q_{t-1} to q in the first scenario and from p to x and from y to q in the second scenario exist in $\hat{H}$ by the induction hypothesis and not by the minimality of $\delta(p,q)$. ∎

We end this section by stating its main Theorem. The proof of this Theorem stems from Lemma 2.4 and Lemma 2.5.

Theorem 2.6. *Let (V,δ) be a metric space of constant doubling dimension with a radius assignment $r(\cdot)$ for its points and let $I = (V,E,r)$ be its corresponding disk graph. Let $I' = (V,E',r_{1+\epsilon})$ be the corresponding disk graph in the relaxed model. It is possible to create a $(1+\epsilon)$-spanner of size $O(n/\epsilon^d)$ for I using edges of I'.*

3. Concluding remarks and open problems

This paper presents a spanner construction for disk graphs in a slightly relaxed model that is as good as spanners for complete graphs and unit disk graphs. This result opens many other research directions for disk graphs. We list here two questions that we find particularly intriguing: Is it possible to design an efficient compact routing scheme for disk graphs? And is it possible to build an efficient distance oracle for disk graphs?

References

[1] U. Zwick D. Dor, S. Halperin. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000.

[2] J. Gao, L. Guibas, and A. Nguyen. Deformable spanners and applications. In *Proc. 20th ACM Symp. on Computational Geometry*, pages 179–199, 2004.

[3] Jie Gao, Leonidas J. Guibas, John Hershberger, Li Zhang, and An Zhu. Geometric spanners for routing in mobile networks. *IEEE J. on Selected Areas in Communications*, 23(1):174–185, 2005.

[4] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Computing*, 35:1148–1184, 2006.

[5] C. Wagner L. Cowen. Compact roundtrip routing for digraphs. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 885–886, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[6] U. Zwick L. Roditty, M. Thorup. Roundtrip spanners and roundtrip routing in directed graphs. *ACM Trans. Algorithms*, 4(3):1–17, 2008.

[7] Xiang-Yang Li, Gruia Calinescu, Peng-Jun Wan, and Yu Wang. Localized delaunay triangulation with application in ad hoc wireless networks. *IEEE Trans. on Parallel and Distributed Systems*, 14(10):1035–1047, 2003.

[8] D. Peleg and L. Roditty. Localized spanner construction for ad hoc networks with variable transmission range. In *Proc. 7th Int. Conf. on Ad-Hoc Networks and Wireless (AdHoc-NOW)*, pages 622–633, 2008.

[9] L. Roditty. Fully dynamic geometric spanners. symposium on computational geometry. pages 373–380, 2007.

[10] Yu Wang and Xiang-Yang Li. Efficient delaunay-based localized routing for wireless sensor networks. *Int. J. of Communication Systems*, 20(7):767–789, 2006.

[11] Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 621-632
www.stacs-conf.org

UNSATISFIABLE LINEAR CNF FORMULAS ARE LARGE AND COMPLEX

DOMINIK SCHEDER

ETH Zürich, Institute of Theoretical Computer Science
Universitätstrasse 6, CH-8092 Zürich, Switzerland
E-mail address: dscheder@inf.ethz.ch

Abstract. We call a CNF formula *linear* if any two clauses have at most one variable in common. We show that there exist unsatisfiable linear k-CNF formulas with at most $4k^2 4^k$ clauses, and on the other hand, any linear k-CNF formula with at most $\frac{4^k}{8e^2k^2}$ clauses is satisfiable. The upper bound uses probabilistic means, and we have no explicit construction coming even close to it. One reason for this is that unsatisfiable linear formulas exhibit a more complex structure than general (non-linear) formulas: First, any treelike resolution refutation of any unsatisfiable linear k-CNF formula has size at least $2^{2^{\frac{k}{2}-1}}$. This implies that small unsatisfiable linear k-CNF formulas are hard instances for Davis-Putnam style splitting algorithms. Second, if we require that the formula F have a *strict* resolution tree, i.e. every clause of F is used only once in the resolution tree, then we need at least $a^{a^{\cdot^{\cdot^{\cdot^{a}}}}}$ clauses, where $a \approx 2$ and the height of this tower is roughly k.

1. Introduction

How can CNF formulas become unsatisfiable? Roughly speaking, there are two ways: Either some constraint (clause) is itself impossible to satisfy – the empty clause; or, every clause can be satisfied individually, but one cannot satisfy all of them simultaneously. In the latter case, the clauses have to somehow overlap. How much? For example, take k boolean variables $x_1, \ldots, x_k$. The conjunction of all 2^k possible clauses of size k is the *complete k-CNF formula* and denote by $\mathcal{K}_k$. It is unsatisfiable, and as small as possible: Any k-CNF formula with less than 2^k clauses is satisfiable. Clearly, the clauses of $\mathcal{K}_k$ overlap a lot. What if we require that any two distinct clauses share at most one variable? We call such a formula *linear*. There are unsatisfiable linear k-CNF formulas, but they are significantly larger and have a much more complex structure than $\mathcal{K}_k$.

A CNF formula is a conjunction (AND) of *clauses*, and a clause is a disjunction (OR) of *literals*. A literal is either a boolean variable x or its negation $\bar{x}$. We require that a clause does not contain the same literal twice, and does not contain complementary literals, i.e.,

1998 ACM Subject Classification: Computational and structural complexity.
Key words and phrases: Extremal Combinatorics, Proof Complexity, Probabilistic Method.
Research is supported by the SNF Grant 200021-118001/1.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2490

both x and $\bar{x}$. To simplify notation, we also regard formulas as sets of clauses and clauses as sets of literals. A clause with k literals is a k-clause, and a k-CNF formula is a CNF formula consisting of k-clauses. For a clause C, we denote by $\mathrm{vbl}(C)$ set of variables x with $x \in C$ or $\bar{x} \in C$. Consequently, a CNF formula F is *linear* if $|\mathrm{vbl}(C) \cap \mathrm{vbl}(D)| \leq 1$ for any two distinct clauses $C, D \in F$. As a relaxation of this notion, we call F *weakly linear* if $|C \cap D| \leq 1$ for any distinct $C, D \in F$.

Example. The formula $(\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (\bar{x}_4 \vee \bar{x}_1)$ is linear, whereas $(\bar{x}_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (x_2 \vee x_3)$ is weakly linear, but not linear, and finally $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3)$ is not weakly linear (and not linear, either).

It is not very difficult to construct an unsatisfiable linear 2-CNF formula, but significantly more effort is needed for a 3-CNF formula. It is not obvious whether unsatisfiable linear k-CNF formulas exist for every k. These questions have been asked first by Porschen, Speckenmeyer and Randerath [15], who also proved that for any $k \geq 3$, if an unsatisfiable linear k-CNF formula exists, then deciding satisfiability of linear k-CNF formulas is NP-complete. Later, Porschen, Speckenmeyer and Zhao [16] and, independently, myself [18] gave a construction of unsatisfiable linear k-CNF formulas, for every $k \in \mathbb{N}_0$:

Theorem 1.1 ([16], [18]). *For every $k \geq 0$, there exists an unsatisfiable linear k-CNF formula F_k, with F_0 containing one clause and F_{k+1} containing $|F_k|2^{|F_k|}$ clauses.*

The $|F_k|$ are extremely large. Here, we will give an almost optimal construction.

Theorem 1.2. *All weakly linear k-CNF formulas with at most $\frac{4^k}{8e^2(k-1)^2}$ clauses are satisfiable. There exists an unsatisfiable linear k-CNF formula with $4k^2 4^k$ clauses.*

It is a common phenomenon in extremal combinatorics that by probabilistic means one can show that a certain object exists (in our case, a "small" linear unsatisfiable k-CNF formula), but one cannot explicitly construct it. We have no explicit construction avoiding the tower-like growth in Theorem 1.1. We give some arguments why this is so, and show that small linear unsatisfiable k-CNF formulas have a more complex structure than their non-linear relatives. To do so, we speak about *resolution*.

1.1. Resolution Trees

If C and D are clauses and there is unique literal u such that $u \in C$ and $\bar{u} \in D$, then $(C \setminus \{u\}) \cup (D \setminus \{\bar{u}\})$ is called the *resolvent* of C and D. It is easy to check that every assignment satisfying C and D also satisfies the resolvent.

Definition 1.3. A *resolution tree* for a CNF formula F is a tree T whose vertices are labeled with clauses, such that

- each leaf of T is labeled with a clause of F,
- the root of T is labeled with the empty clause,
- if vertex a has children b and c, and these are labeled with clauses C_a, C_b, C_c, respectively, then C_a is the resolvent of C_b and C_c.

It is well-known that a CNF formula F is unsatisfiable if and only if it has a resolution tree (which can be exponentially large in $|F|$). Proving lower bounds on the size of resolution trees (and general resolution proofs, which we will not introduce here) has been and still is an area of intensive research. See for example Ben-Sasson and Wigderson [2].

Theorem 1.4. *Let $k \geq 2$. Every resolution tree of an unsatisfiable weakly linear k-CNF formula has at least $2^{2^{\frac{k}{2}-1}}$ leaves.*

A large ratio between the size of F and the size of a smallest resolution tree is an indication that F has a complex structure. For example, it is well-known that the running time of so-called Davis-Putnam procedures on a formula F is lower bounded by the size of the smallest resolution tree of F (actually those procedures were introduces by Davis, Logeman and Loveland [3]). Such a procedure tries to find a satisfying assignment for a formula F (or to prove that none exists) by choosing a variable x, and then recursing on the formulas $F^{[x \mapsto 0]}$ and $F^{[x \mapsto 1]}$, obtained from F by fixing the value of x to 0 or 1, respectively. If F is unsatisfiable, the procedure implicitly constructs a resolution tree.

A CNF formula F is *minimal unsatisfiable* if it is unsatisfiable, and for every clause $C \in F$, $F \setminus \{C\}$ is satisfiable. The complete k-CNF formula introduced above is minimal unsatisfiable, and has a resolution tree with 2^k leaves, one for every clause. This is as small as possible, since for a minimal unsatisfiable formula, every clause must appear as label of at least one leaf of any resolution tree. We call a resolution tree *strict* if no two leaves are labeled by the same clause, and a formula F *strictly treelike* if it has a strict resolution tree. In some sense, strictly treelike formulas are the least complex formulas possible. For example, the complete formula $\mathcal{K}_k$ and the formulas constructed in the proof of Theorem 1.1 are strictly treelike.

Theorem 1.5. *For any $\epsilon > 0$, there exists a constant c such that for any $k \in \mathbb{N}$, any strictly treelike weakly linear k-CNF formula has at least $\mathrm{tower}_{2-\epsilon}(k-c)$ clauses, where $\mathrm{tower}_a(n)$ is defined by $\mathrm{tower}_a(0) = 1$ and $\mathrm{tower}_a(n+1) = a^{\mathrm{tower}_a(n)}$.*

Strictly treelike formulas appear in other contexts, too. Consider MU(1), the class of minimal unsatisfiable formulas whose number of variables is one less than the number of clauses. A result of Davydov, Davydova and Kleine Büning ([4], Theorem 12) implies that every MU(1)-formula is strictly treelike. Also, MU(1)-formulas serve as "universal patterns" for unsatisfiable formulas: Szeider [19] shows that a formula F is unsatisfiable if and only if it can be obtained from a MU(1)-formula G by renaming the variables of G (in a possibly non-injective manner). It is not difficult to show that a strictly treelike linear k-CNF formula can be transformed into a linear MU(1)-formula with the same number of clauses.

1.2. Related Work

For a CNF formula F and a variable x, let $d_F(x)$ denote the *degree of* x, i.e. the number of clauses of F containing x or $\bar{x}$, and let $d(F) := \max_x d_F(x)$ denote the *maximum degree* of F. For the complete k-CNF formula $\mathcal{K}_k$, we have $d(\mathcal{K}_k) = 2^k$. Intuitively, in an unsatisfiable k-CNF formula, some variables should occur in many clauses. In other words, the following function should be large:

$$f(k) := \max\{d \mid \text{every k-CNF formula } F \text{ with } d(F) \leq d \text{ is satisfiable}\} \ . \qquad (1.1)$$

The function $f(k)$ has first been investigated by Tovey [20], who showed $f(k) \geq k$,using Hall's Theorem. Using the famous Lovász Local Lemma (see [5] for the original proof, or [1] for several generalized versions), Kratochvíl, Savický and Tuza [12] proved that $f(k) \geq \frac{2^k}{ek}$, and that while all k-CNF formulas F with $d(F) \leq f(k)$ are trivially satisfiable, deciding satisfiability of k-CNF formulas F with $d(F) \leq f(k)+1$ is already NP-complete, for $k \geq 3$. For $k=3$, this is already observed in [20]. For an upper bound, the complete k-CNF formula witnesses that $f(k) \leq 2^k - 1$. Savický and Sgall [17] showed $f(k) \in O(k^{-0.26}2^k)$. This was improved by Hoory and Szeider [9] to $f(k) \in O\left(\frac{\ln(k)2^k}{k}\right)$, and recently Gebauer [7] proved that $f(k) \leq \frac{2^{k+2}}{k}$. Thus, $f(k)$ is known up to a constant factor. The best upper bounds on $f(k)$ come from MU(1)-formulas. This is true for large values of k, since the formulas constructed in [7] are MU(1), as for small values: Hoory and Szeider [8] show that the function $f(k)$, when restricted to MU(1)-formulas, is computable (in general this is not known), and derive the currently best-known bounds on $f(k)$ for small k ($k \leq 9$). To summarize: When we try to find unsatisfiable k-CNF formulas minimizing a certain parameter, like number of clauses or maximum degree, strictly treelike formulas do an excellent job. However, if we try to construct a small unsatisfiable linear k-CNF formula, they perform horribly. Just compare our upper bound in Theorem 1.2 with the lower bound for strictly treelike formulas in Theorem 1.5

While interest in linear CNF formulas is rather young, *linear hypergraphs* have been studied for quite some time. A hypergraph $H = (V, E)$ is linear if $|e \cap f| \leq 1$ for any two distinct hyperedges $e, f \in E$. A k-uniform hypergraph is a hypergraph where every hyperedge has cardinality k. We ask when a hypergraph 2-colorable, i.e., admits a 2-coloring of its vertices such that no hyperedge becomes monochromatic. Bounds on the number of edges in such a hypergraph were given by Erdős and Lovász [5] (interestingly, this is the paper where the Local Lemma has been proven). They show that there are non-2-colorable linear k-uniform hypergraphs with $ck^4 4^k$ hyperedges, but not with less than $\frac{c' 4^k}{k^3}$. The proof of the lower bound directly translates into our lower bound for linear k-CNF formulas. For the number of edges in linear k-uniform hypergraphs that are not 2-colorable, the currently best upper bound is $ck^2 4^k$ by Kostochka and Rödl [11], and the best lower bound is $k^{-\epsilon}4^k$, for any $\epsilon > 0$ and sufficiently large k, due to Kostochka and Kumbhat [10].

2. Existence and Upper and Lower Bounds

Proof of Theorem 1.1. Choose F_0 to be the formula consisting of only the empty clause. Suppose we have constructed F_k, and want to construct F_{k+1}. Let $m = |F_k|$. We create m new variables $x_1, \ldots, x_m$, and let $\mathcal{K}_m = \{D_1, D_2, \ldots, D_{2^m}\}$ be the complete m-CNF formula over $x_1, \ldots, x_m$. It is unsatisfiable, but not linear. We take 2^m variable disjoint copies of F_k, denoted by $F_k^{(1)}, F_k^{(2)}, \ldots, F_k^{(2^m)}$. For each $1 \leq i \leq 2^m$, we build a linear $(k+1)$-CNF formula $\tilde{F}_k^{(i)}$ from $F_k^{(i)}$ by adding, for each $1 \leq j \leq m$, the j^{th} literal of D_i to the j^{th} clause of $F_k^{(i)}$. Note that every assignment satisfying $\tilde{F}_k^{(i)}$ also satisfies D_i. Finally, we set $F_{k+1} := \bigcup_{i=1}^{2^m} \tilde{F}_k^{(i)}$. This is an unsatisfiable linear $(k+1)$-CNF formula with $m2^m$ clauses. ■

Using induction, it is not difficult to see that the formulas F_k are strictly treelike. We will prove the upper bound in Theorem 1.2 by giving a probabilistic construction of a comparably small unsatisfiable linear k-CNF formula. Our construction consists of two steps. First, we construct a linear k-uniform hypergraph H that is "dense" in the sense that $\frac{m}{n}$ is large, where m and n are the number of hyperedges and vertices, respectively, and then transform it randomly into a linear k-CNF formula F that is unsatisfiable with high probability.

Lemma 2.1. *If there is a linear k-uniform hypergraph H with n vertices and m edges such that $\frac{m}{n} \geq 2^k$, then there is an unsatisfiable linear k-CNF formula with m clauses.*

Proof. Let $H = (V, E)$. By viewing V as a set of variables and E as a set of clauses (each containing only positive literals), this is a (satisfiable) linear k-CNF formula. We replace each literal in each clause by its complement with probability $\frac{1}{2}$, independently in each clause. Let F denote the resulting (random) formula. For any fixed truth assignment α, it holds that $\Pr[\alpha \text{ satisfies } F] = (1 - 2^{-k})^m$. Hence the expected number of satisfying assignments of F is

$$2^n(1 - 2^{-k})^m < 2^n e^{-2^{-k}m} = e^{\ln(2)n - 2^{-k}m} \leq 1 \ ,$$

where the last inequality follows from $\frac{m}{n} \geq 2^k$. Hence some formula F has fewer than one satisfying assignment, i.e., none. ■

How can we construct a dense linear hypergraph? We use a construction by Kuzjurin [13]. Our application of this construction is motivated by Kostochka and Rödl [11], who use it to construct linear hypergraphs of large chromatic number.

Lemma 2.2. *For any prime power q and any $k \in \mathbb{N}$, there exists a k-uniform linear hypergraph with kq vertices and q^2 edges.*

With $n = kq$, this hypergraph has n^2/k^2 hyperedges. This is almost optimal, since any linear k-uniform hypergraph on n vertices has at most $\binom{n}{2}/\binom{k}{2}$ hyperedges: The n vertices provide us with $\binom{n}{2}$ vertex pairs. Each hyperedge occupies $\binom{k}{2}$ pairs, and because of linearity, no pair can be occupied by more than one hyperedge.

Proof. Choose the vertex set $V = V_1 \uplus \cdots \uplus V_k$, where each V_i is a disjoint copy of the finite field $GF(q)$. The hyperedges consist of all k-tuples $(x_1, \ldots, x_k)$ with $x_i \in V_i, 1 \leq i \leq k$, such that

$$\begin{pmatrix} 1 & 1 & & 1 & & 1 \\ 1 & 2 & \ldots & i & \ldots & k \\ 1 & 4 & & i^2 & & k^2 \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & 2^{k-3} & \ldots & i^{k-3} & \ldots & k^{k-3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_k \end{pmatrix} = \mathbf{0} \ . \tag{2.1}$$

Consider two distinct vertices $x \in V_i$, $y \in V_j$. How many hyperedges contain both of them? If $i = j$, none. If $i \neq j$, we can find out by plugging the fixed values x, y into (2.1). We obtain a (possibly non-uniform) $(k-2) \times (k-2)$ linear system with a Vandermonde matrix, which has a unique solution. In other words, x and y are in exactly one hyperedge, and the hypergraph is linear. By the same argument, there are exactly q^2 hyperedges. ■

Proof of the upper bound in Theorem 1.2. Choose a prime power $q \in \{k2^k, \dots, 2k2^k - 1\}$. By Lemma 2.2, there is a linear k-uniform hypergraph H with $n = qk$ vertices and $m = q^2$ hyperedges. Since $\frac{m}{n} = \frac{q}{k} \geq 2^k$, Lemma 2.1 shows that there is an unsatisfiable linear k-CNF formula with $q^2 \leq 4k^2 4^k$ clauses. ∎

Let us prove the lower bound of Theorem 1.2. For a literal u and a CNF formula F, we write $\mathrm{occ}_F(u) := |\{C \in F \mid u \in C\}|$, the *degree* of the literal u. Thus $d_F(x) = \mathrm{occ}_F(x) + \mathrm{occ}_F(\bar{x})$. We write $\mathrm{occ}(F) = \max_u \mathrm{occ}_F(u)$. In analogy to $f(k)$, we define $f_{\mathrm{occ}}(k)$ to be the largest integer d such that any k-CNF formula F with $\mathrm{occ}(F) \leq d$ is satisfiable. Clearly $f_{\mathrm{occ}}(k) \geq \frac{f(k)}{2}$, and thus from [12] it follows that $f_{\mathrm{occ}}(k) \geq \frac{2^k}{2ek}$. Actually, an application of the *Lopsided* Lovász Local Lemma [6, 1, 14] yields $f_{\mathrm{occ}}(k) \geq \frac{2^k}{ek} - 1$.

Lemma 2.3. *Let F be a linear k-CNF formula with at most $1 + f_{\mathrm{occ}}(k-1)$ variables of degree at least $1 + f_{\mathrm{occ}}(k-1)$. Then F is satisfiable.*

Proof. Transform F into a $(k-1)$-CNF formula F' by removing in every clause in F a literal of maximum degree. We claim that $\deg_{F'}(u) \leq f_{\mathrm{occ}}(k-1)$ for every literal u. Therefore F' is satisfiable, and F is, as well.

For the sake of contradiction, suppose there is a literal u such that $t := \mathrm{occ}_{F'}(u) \geq 1 + f_{\mathrm{occ}}(k-1)$. Let C'_i, $i = 1, 2, \dots, t$, be the clauses in F' containing u. C'_i is obtained by removing some literal v_i from some clause $C_i \in F$. By construction of F', $\mathrm{occ}_F(v_i) \geq \mathrm{occ}_F(u) \geq f_{\mathrm{occ}}(k-1) + 1$ for all $1 \leq i \leq t$. The v_i are pairwise distinct: If $v_i = v_j$, then $\{u, v_i\} \subseteq C_i \cap C_j$. Since F is weakly linear, this can only mean $i = j$. Now $u, v_1, v_2, \dots, v_t$ are $t + 1 \geq 2 + f_{\mathrm{occ}}(k-1)$ variables of degree at least $1 + f_{\mathrm{occ}}(k-1)$ in F, a contradiction. ∎

We see that an unsatisfiable weakly linear k-CNF formula has at least $f_{\mathrm{occ}}(k-1) + 2 \geq \frac{2^k}{2e(k-1)} + 1$ literals of degree at least $f_{\mathrm{occ}}(k-1) + 1 \geq \frac{2^k}{2e(k-1)}$. Double counting yields $k|F| = \sum_u \mathrm{occ}_F(u) > \frac{4^k}{4e^2(k-1)^2}$, thus $|F| > \frac{4^k}{16e^2k^3}$. By a more careful argument, we can improve this by a factor of k. We call a hypergraph (j, d)*-rich* if at least j vertices have degree at least d. The following lemma is due to Welzl [22].

Lemma 2.4. *For $d \in \mathbb{N}_0$, every linear (d, d)-rich hypergraph has at least $\binom{d+1}{2}$ edges. This bound is tight for all $d \in \mathbb{N}_0$.*

Proof. We proceed by induction over d. Clearly, the assertion of the lemma is true for $d = 0$. Now let $H = (V, E)$ be a linear (d, d)-rich hypergraph for $d \geq 1$. Choose some vertex v of degree at least d in H and let $H' = (V, E')$ be the hypergraph with $E' := E \setminus \{e \in E \mid e \ni v\}$. We have (i) $|E| \geq |E'| + d$, (ii) H' is linear, since this property is inherited when edges are removed, and (iii) H' is $(d-1, d-1)$-rich, since for no vertex other than v the degree decreases by more than 1 due to the linearity of H. It follows hat $|E| \geq \binom{d}{2} + d = \binom{d+1}{2}$. The complete 2-uniform hypergraph (graph, so to say) on $d + 1$ vertices shows that the bound given is tight for all $d \in \mathbb{N}_0$. ∎

Proof of the lower bound in Theorem 1.2. A weakly linear k-CNF formula F is a linear k-uniform hypergraph, with literals as vertices. If F is unsatisfiable, then by Lemma 2.3, it is $(f_{\mathrm{occ}}(k-1) + 1, f_{\mathrm{occ}}(k-1) + 1)$-rich. By Lemma 2.4, F has at least $\binom{f_{\mathrm{occ}}(k-1)+2}{2} > \frac{4^k}{8e^2(k-1)^2}$ clauses. ∎

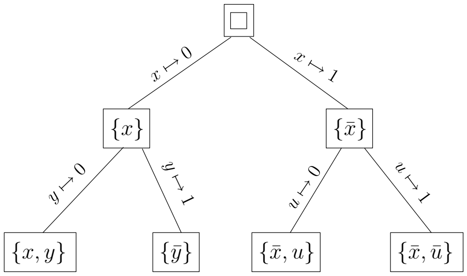


Figure 1: A resolution tree, with its edges labeled in the obvious way. Every clause is unsatisfied when applying the assignments on the path to the root.

There is an obvious generalization of the notion of being linear. We say a CNF formula is *b-linear*, if any two distinct clauses $C, D \in F$ fulfill $|\text{vbl}(C) \cap \text{vbl}(D)| \leq b$, and *weakly b-linear* if $|C \cap D| \leq b$ holds for all distinct $C, D \in F$. Thus, a (weakly) 1-linear formula is (weakly) linear. We can generalize Theorem 1.2 for $b \geq 2$. However, the proofs do not introduce new ideas and goes along the lines of the proofs presented above.

Theorem 2.5. *Let $b \geq 2$. Every weakly b-linear k-CNF formula with at most $\frac{2^{k(1+\frac{1}{b})}}{2^{b+2}e^2k^{2+\frac{1}{b}}}$ clauses is satisfiable. There exists an unsatisfiable b-linear k-CNF formula with at most $2^{b+1}(k2^k)^{1+\frac{1}{b}}$ clauses.*

3. Proof of Theorem 1.4

Let F be an unsatisfiable weakly linear k-CNF formula, and let T be a resolution tree of minimal size of F. We want to show that T has a large number of nodes. It is not difficult to see that a resolution tree of minimal size is *regular*, meaning that no variable is resolved more than once on a path from a leaf to the root. See Urquhart [21], Lemma 5.1, for a proof of this fact. We take a random walk of length ℓ in T starting at the root, in every step choosing randomly to go to one of the two children of the current node. If we arrive at a leaf, we stay there. We claim that if $\ell \leq \sqrt{2^{k-2}}$, then with probability at least $\frac{1}{2}$, our walk does not end at a leaf. Thus, T has at least $2^{\ell-1}$ inner vertices at distance ℓ from the root, thus at least $2^{2^{\frac{k}{2}-1}}$ leaves.

As illustrated in Figure 1, we label each edge in T with an assignment. If C is the resolvent of D_1 and D_2, $x \in D_1$ and $\bar{x} \in D_2$, we label the edge from C to D_1 by $x \mapsto 0$ and from C to D_2 by $x \mapsto 1$. Each path from the root to a node gives a partial assignment α. If that node is labeled with clause C, then C evaluates to `false` under α. In our random walk, let α_i denote the partial assignment associated with the first i steps. α_0 is the empty assignment, and α_i assigns exactly i variables (if we are not yet at a leaf). We set $F_i := F^{[\alpha_i]}$, i.e., the formula obtained from F by fixing the variables according to the partial assignment α_i. For a formula G, we define the *weight* $w(G)$ to be

$$w(G) := \sum_{C \in G, |C| \leq k-2} 2^{k-|C|} \, . \tag{3.1}$$

Since F is a k-CNF formula, $w(F) = 0$. If some formula G contains the empty clause, then $w(G) \geq 2^k$. In our random walk, $w(F_i)$ is a random variable.

Lemma 3.1. $\mathbb{E}[w(F_{i+1})] \leq \mathbb{E}[w(F_i)] + 4i$.

Since $w(F_0) = 0$, this implies $\mathbb{E}[w(F_\ell)] \leq 4\binom{\ell}{2} \leq 2\ell^2$. If our random walk ends at a leaf, then F_ℓ contains the empty clause, thus $w(F_\ell) \geq 2^k$. Therefore $2\ell^2 \geq \mathbb{E}[w(F_\ell)] \geq 2^k \Pr[\text{the random walk ends at a leaf}]$. We conclude that at least half of all paths of length $\ell^* = \sqrt{2^{k-2}}$ starting at the root do not end at a leaf. Thus T has at least 2^{ℓ^*-1} internal nodes at distance ℓ^* from the root, and thus at least 2^{ℓ^*} leaves, which proves the theorem. It remains to prove the lemma.

Proof of the lemma. For a formula G and a variable x, let $d_{k-1}(x, G)$ denote the number of $(k-1)$-clauses containing x or $\bar{x}$. Since F_0 is a k-CNF formula, $d_{k-1}(x, F_0) = 0$, for all variables x. We claim that $d_{k-1}(x, F_{i+1}) \leq d_{k-1}(x, F_i) + 2$ for every variable x. To see this, note that in step i, some variable y is set to $b \in \{0, 1\}$, say to 0. At most one k-clause of F_i contains y and x, and at most one contains y and $\bar{x}$, since F_i is weakly linear, thus $d_{k-1}(x, F_{i+1}) \leq d_{k-1}(x, F_i) + 2$. It follows immediately that $d_{k-1}(x, F_i) \leq 2i$.

Consider $w(F_i)$, which was in (3.1). F_{i+1} is obtained from F_i by setting some variable y randomly to 0 or 1. Consider a clause C. How does its contribution to (3.1) change when setting y? If (i) $y \notin \text{vbl}(C)$ or $|C| = k$, it does not change. If (ii) $y \in \text{vbl}(C)$ and $|C| \leq k-2$, then with probability $\frac{1}{2}$ each, its contribution to (3.1) doubles or vanishes. Hence on expectation, it does not change. If (iii) $y \in \text{vbl}(C)$ and $|C| = k - 1$, then C contributes nothing to $w(F_i)$, and with probability $\frac{1}{2}$, it contributes 4 to $w(F_{i+1})$. On expectation, its contribution to (3.1) increases by 2. Case (iii) applies to at most $d_{k-1}(y, F_i) \leq 2i$ clauses. Hence $\mathbb{E}[w(F_{i+1})] \leq \mathbb{E}[w(F_i)] + 4i$. ■

4. Proof of Theorem 1.5

Let F be a strictly treelike weakly linear k-CNF formula F, and let T be a strict resolution tree of F. Letters a, b, c denote nodes of T, and u, v, w denote literals. Every node a of T is labeled with a clause C_a. We define a graph G_a with vertex set C_a, connecting $u, v \in C_a$ if $u, v \in D$ for some clause $D \in F$ that occurs as a label of a leaf in the subtree of a. Since T is a strict resolution tree and F is weakly linear, every edge in G_a comes from a unique leaf of T. Resolution now has a simple interpretation as a "calculus on graphs", see Figure 2. If a is a leaf, then $G_a = K_k$. Since the root of a resolution tree is labeled with the empty clause, we have $G_{\text{root}} = (\emptyset, \emptyset)$, For a graph G, let $\kappa_i(G)$ denote the minimum size of a set $U \subseteq V(G)$ such that $G - U$ contains no i-clique. Here, $G - U$ is the subgraph of G induced by $V(G) \setminus U$. Thus, $\kappa_1(G) = |V(G)|$, and $\kappa_2(G)$ is the size of a minimum vertex cover of G. For the complete graph K_k, $\kappa_i(K_k) = k - i + 1$. We write $\kappa_i(a) := \kappa_i(G_a)$. The tuple $(\kappa_1(a), \ldots, \kappa_k(a))$ can be viewed as the complexity measure for a. We observe that if a is a leaf, then $\kappa_i(a) = k - i + 1$, and $\kappa_i(\text{root}) = 0$, for all $1 \leq i \leq k$. If a is an ancestor of b in T, let $\text{dist}(a, b)$ denote the number of edges in the T-path from a to b. Since one resolution step deletes one literal (and may add several), the next proposition is immediate:

Proposition 4.1. *If b is a descendant of a in T, then $\kappa_i(b) \leq \kappa_i(a) + \text{dist}(a, b)$.*

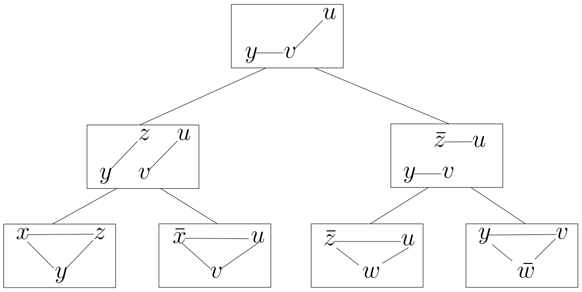


Figure 2: Resolution as a calculus on graphs. A resolution step amounts to deleting the resolved vertex and taking the union of the two graphs.

At this point we want to give an intuition of the proofs that follow. Our goal is to show that if the values $\kappa_i(a)$ are small for some node a in the tree, then the subtree of a is big. The proof goes roughly as follows: If the subtree of a is small, then there are many descendants b of a that are not too far from a and have even smaller subtrees. By induction, we will be able to show that $\kappa_{i+1}(b)$ is fairly large. Thus, on the path from b to a, not all $(i+1)$-cliques are destroyed, and every such descendant b of a provides G_a with an $(i+1)$-clique. These cliques need not be vertex-disjoint, but they are edge-disjoint. This implies that G_a has many vertex-disjoint i-cliques, a contradiction to $\kappa_i(a)$ being small. To make this intuition precise, we have to define what small and big actually means in this context: We fix a value $1 \leq \ell \leq k$ and define ν_i and θ_i for $1 \leq i \leq \ell$ as follows: $\theta_\ell := \left\lfloor \frac{k-\ell+1}{2} \right\rfloor - 1$ and $\nu_\ell := 1$, and for $1 \leq i < \ell$, we inductively define $\theta_i := \left\lfloor \frac{2^{\nu_{i+1}\theta_{i+1}-2}}{\theta_{i+1}} \right\rfloor - 1$ and $\nu_i := \frac{\nu_{i+1}\theta_{i+1}-1}{\theta_i} \left\lfloor \frac{\theta_i}{\theta_{i+1}} \right\rfloor$. One should not worry about these ugly expressions too much, they are only chosen that way to make the induction go through. For the right value of ℓ, one checks that θ_1 is a tower function in k. More precisely, for any $\epsilon > 0$, there exists a $c \in \mathbb{N}$ such that when choosing $\ell = k - c$, then $\theta_1 \geq \text{tower}_{2-\epsilon}(k-c)$. The following theorem is a more precise version of Theorem 1.5.

Theorem 4.2. *Let F be a strictly treelike linear k-CNF formula. Then F has at least $2^{\nu_1\theta_1}$ clauses.*

Proof. A node a in T is *i-extendable* if $\kappa_j(a) \leq \theta_j$ for each $i \leq j \leq \ell$. We observe that if a is i-extendable, it is also $(i+1)$-extendable. For $i = \ell + 1$, the condition is void, so every node is $(\ell+1)$-extendable. Also, the root is 1-extendable, since $\kappa_1(\text{root}) = 0$.

Definition 4.3. A set A of descendants of a in T such that (i) no vertex in A is an ancestor of any other vertex in A and (ii) $\text{dist}(a, b) \leq d$ for all $b \in A$ is called an *antichain of a at distance at most d*. If furthermore every $b \in A$ is i-extendable, we call A an i-extendable antichain.

Lemma 4.4. *Let $1 \leq i \leq \ell$, and let a be a node in T. If a is i-extendable, then there is an $(i+1)$-extendable antichain A of a at distance at most θ_i such that $|A| = 2^{\nu_i\theta_i}$.*

Proof. We use induction on $\ell - i$. For the base case $i = \ell$, we have $\kappa_\ell(a) \leq \theta_\ell$, as a is ℓ-extendable. Since each leaf b of T has $\kappa_\ell(b) = k - \ell + 1 \geq 2\theta_\ell + 2$, Proposition 4.1 tells us that every leaf in the subtree of a has distance at least $\theta_\ell + 2$ from a. Since T is a complete binary tree, there are 2^{θ_ℓ} descendants of a at distance exactly θ_ℓ from a. This is the desired antichain A of a. Since every node is $(\ell+1)$-extendable, the base case holds. For the step,

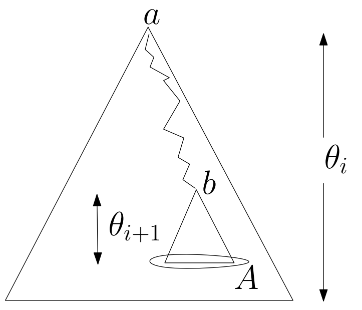


Figure 3: Illustration of the claim in the proof of Lemma 4.4. If node a is i-extendable, and b is a close $(i+1)$-extendable descendant of a, then b itself has many close descendants A, at least half of which are $(i+1)$-extendable themselves.

let a be i-extendable, for $1 \leq i < \ell$.

Claim: Let b be a descendant of a with $\mathrm{dist}(a,b) \leq \theta_i$. If b is $(i+1)$-extendable, then there is an $(i+1)$-extendable antichain A of b at distance at most θ_{i+1} of size $2^{\nu_{i+1}\theta_{i+1}-1}$.

Proof of the claim. By applying the induction hypothesis of the lemma to b, there is an $(i+2)$-extendable antichain A of b at distance at most θ_{i+1} of size $2^{\nu_{i+1}\theta_{i+1}}$. Let $A_{\text{good}} := \{c \in A \mid \kappa_{i+1}(c) \leq \theta_{i+1}\}$. This is an $(i+1)$-extendable antichain. If A_{good} contains at least half of A, we are done. See Figure 3 for an illustration. Write $A_{\text{bad}} := A \setminus A_{\text{good}}$ and suppose for the sake of contradiction that $A_{\text{bad}} > 2^{\nu_{i+1}\theta_{i+1}-1}$. Consider any $c \in A_{\text{bad}}$. On the path from c to b, in each step some literal gets removed (and others may be added). Let P denote the set of the removed literals. Then $C_c \setminus \{P\} \subseteq C_b$, and $G_c - P$ is a subgraph of G_b. Node c is not $(i+1)$-extendable, thus $\kappa_{i+1}(c) \geq \theta_{i+1} + 1$. Since $|P| = \mathrm{dist}(b,c) \leq \theta_{i+1}$, the graph $G_c - P$ contains at least one $(i+1)$-clique, which is also contained in G_b. This holds for every $c \in A_{\text{bad}}$, and by weak linearity, G_b contains at least $|A_{\text{bad}}|$ edge disjoint $(i+1)$-cliques. Since b is $(i+1)$-extendable, there exists a set $U \subseteq V(G_v), |U| = \kappa_{i+1}(b)$ such that $G_b - U$ contains no $(i+1)$-clique. Each of the $|A_{\text{bad}}|$ edge-disjoint $(i+1)$-cliques G_b contains some vertex of U, thus some vertex $v \in U$ is contained in at least $\frac{|A_{\text{bad}}|}{|U|} \geq \frac{2^{\nu_{i+1}\theta_{i+1}-1}}{\theta_{i+1}} \geq 2\theta_i + 1$ edge-disjoint $(i+1)$-cliques. Two such cliques overlap in no vertex besides v, hence G_b contains at least $2\theta_i + 1$ *vertex-disjoint* i-cliques, thus $\kappa_i(b) \geq 2\theta_i + 1$. By Proposition 4.1, $\kappa_i(a) \geq \kappa_i(b) - \mathrm{dist}(a,b) \geq \theta_i + 1$. This contradicts the assumption of Lemma 4.4 that a is i-extendable. We conclude that $|A_{\text{bad}}| \leq \frac{1}{2}|A|$, which proves the claim. ∎

Let us continue with the proof of the lemma. If A is an $(i+1)$-extendable antichain of a at distance $d \leq \theta_i$, then by the claim for each vertex $b \in A$ there exists an $(i+1)$-extendable antichain of b at distance at most θ_{i+1}, of size $2^{\nu_{i+1}\theta_{i+1}-1}$. Their union is an $(i+1)$-extendable antichain A' of a at distance at most $d + \theta_{i+1}$, of size $|A|2^{\nu_{i+1}\theta_{i+1}-1}$. Hence we can "inflate" A to A', as long as $d \leq \theta_i$. Starting with the $(i+1)$-extendable antichain $\{a\}$ and inflate it $\left\lfloor \frac{\theta_i}{\theta_{i+1}} \right\rfloor$ times, and obtain a final $(i+1)$-extendable antichain of a at distance at most θ_i of size at least $\left(2^{\nu_{i+1}\theta_{i+1}-1}\right)^{\left\lfloor \frac{\theta_i}{\theta_{i+1}} \right\rfloor} = 2^{\nu_i\theta_i}$. ∎

Applying Lemma 4.4 to the root of T, which is 1-extendable, we obtain an antichain A of size $2^{\nu_1 \theta_1}$ nodes. Since T has at least $|A|$ leaves, this proves the theorem. ■

5. Open Problems

Let $f_{\mathrm{LIN}}(k)$ be the largest integer d such that any linear k-CNF formula F with $d(F) \leq d$ is satisfiable. Clearly $f_{\mathrm{LIN}}(k) \geq f(k)$, and from the proof of the upper bound in Theorem 1.2 it follows that $f_{\mathrm{LIN}}(k) \leq 2k2^k$. Is there a significant gap between $f(k)$ and $f_{\mathrm{LIN}}(k)$? It is not difficult to show that $f(2) = f_{\mathrm{LIN}}(2) = 2$, but we do not know the value of $f_{\mathrm{LIN}}(k)$ for any $k \geq 3$. How do unsatisfiable linear k-CNF formulas look like? Can one find an explicit construction of an unsatisfiable linear k-CNF formula whose size is singly exponential in k? We suspect one has to come up with some algebraic construction. What is the resolution complexity of linear k-CNF formulas? Tree resolution complexity is doubly exponential in k. We suspect the same to be true for general resolution.

Acknowledgments

My thanks go to Emo Welzl, Robin Moser, Heidi Gebauer, Andreas Razen and Philipp Zumstein for very helpful and fruitful discussions.

References

[1] N. Alon and J. H. Spencer. *The probabilistic method.* Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience [John Wiley & Sons], New York, second edition, 2000. With an appendix on the life and work of Paul Erdős.

[2] E. Ben-Sasson and A. Wigderson. Short proofs are narrow—resolution made simple. *J. ACM*, 48(2):149–169, 2001.

[3] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Comm. ACM*, 5:394–397, 1962.

[4] G. Davydov, I. Davydova, and K. Büning. An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. *Ann. Math. Artificial Intelligence*, 23(3-4):229–245, 1998.

[5] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal, R. Rado, and V. T. Sós, editors, *Infinite and Finite Sets (to Paul Erdős on his 60th birthday), Vol. II*, pages 609–627. North-Holland, 1975.

[6] P. Erdős and J. Spencer. Lopsided Lovász Local Lemma and Latin transversals. *Discrete Appl. Math.*, 30(2-3):151–154, 1991. ARIDAM III (New Brunswick, NJ, 1988).

[7] H. Gebauer. Disproof of the neighborhood conjecture with implications to SAT. In A. Fiat and P. Sanders, editors, *17th Annual European Symposium on Algorithms (ESA 2009)*, volume 5757 of *Lecture Notes in Computer Science*, pages 764–775. Springer, 2009.

[8] S. Hoory and S. Szeider. Computing unsatisfiable k-SAT instances with few occurrences per variable. *Theoretical Computer Science*, 337(1-3):347–359, 2005.

[9] S. Hoory and S. Szeider. A note on unsatisfiable k-CNF formulas with few occurrences per variable. *SIAM Journal on Discrete Mathematics*, 20(2):523–528, 2006.

[10] A. V. Kostochka and M. Kumbhat. Coloring uniform hypergraphs with few edges, *manuscript.*

[11] A. V. Kostochka and V. Rödl. Constructions of sparse uniform hypergraphs with high chromatic number, *manuscript.*

[12] J. Kratochvíl, P. Savický, and Z. Tuza. One more occurrence of variables makes satisfiability jump from trivial to NP-complete. *SIAM Journal of Computing*, 22(1):203–210, 1993.

[13] N. N. Kuzjurin. On the difference between asymptotically good packings and coverings. *European J. Combin.*, 16(1):35–40, 1995.

[14] L. Lu and L. Székely. Using Lovász Local Lemma in the space of random injections. *Electron. J. Combin.*, 14(1):Research Paper 63, 13 pp. (electronic), 2007.

[15] S. Porschen, E. Speckenmeyer, and B. Randerath. On linear CNF formulas. In *SAT*, pages 212–225, 2006.

[16] S. Porschen, E. Speckenmeyer, and X. Zhao. Linear CNF formulas and satisfiability. *Discrete Appl. Math.*, 157(5):1046–1068, 2009.

[17] P. Savický and J. Sgall. DNF tautologies with a limited number of occurrences of every variable. *Theoret. Comput. Sci.*, 238(1–2):495–498, 2000.

[18] D. Scheder. Unsatisfiable linear k-CNFs exist, for every k. *CoRR*, abs/0708.2336, 2007.

[19] S. Szeider. Homomorphisms of conjunctive normal forms. *Discrete Appl. Math.*, 130(2):351–365, 2003.

[20] C. A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Appl. Math.*, 8(1):85–89, 1984.

[21] A. Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 1:425–467, 1995.

[22] E. Welzl. personal communication.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 633-644
www.stacs-conf.org

Construction Sequences and Certifying 3-Connectedness

JENS M. SCHMIDT

Dept. of Computer Science, Freie Universität, Berlin, Germany
E-mail address: jens.schmidt@inf.fu-berlin.de

Abstract. Tutte proved that every 3-connected graph on more than 4 nodes has a *contractible edge*. Barnette and Grünbaum proved the existence of a *removable edge* in the same setting. We show that the sequence of contractions and the sequence of removals from G to the K_4 can be computed in $O(|V|^2)$ time by extending Barnette and Grünbaum's theorem. As an application, we derive a certificate for the 3-connectedness of graphs that can be easily computed and verified.

1. Introduction

Instead of dealing with contractions or removals in a 3-connected graph $G = (V, E)$ we take the equivalent view of starting with the complete graph on four vertices K_4 and applying their inverse operations until G is constructed. Such a sequence is called a *construction sequence* of G. We will define contractions, removals and their inverse operations in Section 2.

Although existence theorems on contractible and removable edges are used frequently in graph theory [14, 10, 11], we are not aware of any computational results to find the whole construction sequence, except when contractions and removals are allowed to intermix [1]. Moreover, efficient algorithms are unlikely to be derived from the existence proofs as they, e. g., in the case of Barnette and Grünbaum, depend heavily on adding longest paths, which are NP-hard to find. In contrast, we show that it is possible to find a construction sequence for a graph G in time $O(|V|^2)$ for Barnette and Grünbaum's characterization, at the expense of having parallel edges in intermediate graphs. In addition, we show that Barnette and Grünbaum's sequence can be transformed in linear time to Tutte's sequence of contractions and is therefore algorithmically at least as powerful. Both algorithms do not rely on the 3-connectedness test of Hopcroft and Tarjan [6], which runs in linear time but is rather involved.

Blum and Kannan [3] introduced the concept of *certifying algorithms*, which give an easy-to-verify proof of correctness along with their output. While being important for program verification, certifying algorithms provide often new insights into a problem, which

Key words and phrases: Algorithms and data structures, construction sequence, 3-connected, certifying algorithm, Tutte contraction, removable edges, *ACM classification*: F.2.2;G.2.2.

This research was supported by the Deutsche Forschungsgemeinschaft within the research training group "Methods for Discrete Structures" (GRK 1408).

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2491

can lead to new methods. For that reasons they are a major goal for problems on which the fast solutions known are complicated and difficult to implement. Testing a graph on 3-connectedness is such a problem, but surprisingly few work has been devoted to certifying algorithms, although a sophisticated linear-time algorithm without certificates is known for over 35 years [6, 15, 16]. In fact, we are aware of only one certifying algorithm for that problem [1], which runs in quadratic time, but is quite involved. Using construction sequences, we give a simple, alternative solution with running time $O(|V|^2)$ and show that the used certificate is easy to verify in time $O(|E|)$.

We first recapitulate well-known results on the existence of construction sequences in Sections 2.1 and 2.2 and point out how Tutte's sequence can be obtained from Barnette and Grünbaum's sequence in linear time. Sections 2.3 and 3 cover the main idea for the existence result that we use for computing Barnette and Grünbaum's sequence. Section 4 deals with the question how construction sequences are efficiently represented and Section 5 shows how to use construction sequences for a certifying 3-connectedness test.

2. Construction Sequences

Let $G = (V, E)$ be a finite graph with $n := |V|$, $m := |E|$, $V(G) = V$ and $E(G) = E$. A graph is *connected* if there is a path between any two nodes and *disconnected* otherwise. For $k \geq 1$, a graph is *k-connected* if $n > k$ and deleting every $k-1$ nodes leaves a connected graph. A node (a pair of nodes) that leaves a disconnected graph upon deletion is called a *cut vertex* (a *separation pair*). Note that k-connectedness does not depend on parallel edges nor on self-loops. A path leading from node v to node w is denoted by $v \rightarrow w$. For a node v in a graph, let $N(v) = \{w \mid vw \in E\}$ denote its set of neighbors and $deg(v)$ its degree. For a graph G, let $\delta(G)$ be the minimum degree of its vertices.

A *subdivision* of a graph replaces each edge by a path of length at least one. Conversely, we want a notation to get back to the graph without subdivided edges. If $deg(v) = 2$, $|N(v)| = 2$ and $v \notin N(v)$ for a graph G, let $smooth_v(G)$ be the graph obtained from G by deleting v followed by adding an edge between its neighbors; we say v is *smoothed*. If one of the conditions is violated, let $smooth_v(G) = G$. Let $smooth(G)$ be the graph obtained by smoothing every node in G. For an edge $e \in E$, let $G \setminus e$ denote the graph obtained from G by deleting e. Let K_n be the complete graph on n nodes.

The following are well-known corollaries of Menger's theorem [8].

Lemma 2.1. (Fan Lemma) *Let v be a node in a graph G that is k-connected with $k \geq 1$ and let A be a set of at least k nodes in G with $v \notin A$. Then there are k internally node-disjoint paths $P_1, \ldots, P_k$ from v to distinct nodes $a_1, \ldots, a_k \in A$ such that for each of these paths $V(P_i) \cap A = a_i$.*

Lemma 2.2. (Expansion Lemma [17]) *Let G be a k-connected graph. Then the graph obtained by adding a new node v joined to at least k nodes in G is still k-connected.*

2.1. Tutte's Characterization and their Inverse

From now on we assume for simplicity that our input graph $G = (V, E)$ is simple although all results can be extended to multigraphs. Generally, contractions cannot always avoid parallel edges in intermediate graphs, e. g., for wheels. That is why we define contractions to preserve graphs to be simple: *Contracting* an edge $e = xy$ in a graph deletes e, identifies nodes x and y and replaces iteratively all 2-cycles by an edge. An edge e is called *contractible* if contracting e results in a 3-connected graph.

A *node splitting* takes a node v of a 3-connected graph, replaces v by two nodes x and y with an edge between them and replaces every former edge uv that was incident to v with either the edge ux, uy or both such that $|N(x)| \geq 3$ and $|N(y)| \geq 3$ in the new graph. Node splitting as defined here is therefore the exact inverse of contracting a contractible edge that has on both endnodes at least 3 neighbors.

Theorem 2.3. (Corollary of Tutte [13]) *The following statements are equivalent:*

$$\begin{aligned} & \text{A simple graph } G \text{ is 3-connected} \\ \Leftrightarrow\ & \exists \text{ sequence of contractions from } G \text{ to } K_4 \text{ on contractible edges } e = xy \\ & \text{with } |N(x)| \geq 3 \text{ and } |N(y)| \geq 3 && (2.1) \\ \Leftrightarrow\ & \exists \text{ construction sequence from } K_4 \text{ to } G \text{ using node splittings} && (2.2) \end{aligned}$$

We describe next a straight-forward $O(n^2)$ algorithm to compute (2.1) for a graph G on more than 4 vertices. First, we decrease the number of edges to $O(n)$ in G by applying the algorithm of Nagamochi and Ibaraki [9]. This preserves the 3-connectedness or respectively, the non 3-connectedness of G. Moreover, it is known that the resulting graph contains a vertex v of degree 3. By a result of Halin [5], every node of degree 3 is incident to a contractible edge e. We get e by subsequently contracting each of the three incident edges and testing the resulting graph with the algorithm of Hopcroft and Tarjan [6] for 3-connectedness. Iteration of both subroutines gives us the whole contraction sequence in $O(n^2)$ time. However, the Hopcroft-Tarjan test is difficult to implement and we will give a much simpler algorithm that is capable of computing both characterizations later.

2.2. Barnette and Grünbaum's Characterization and their Inverse

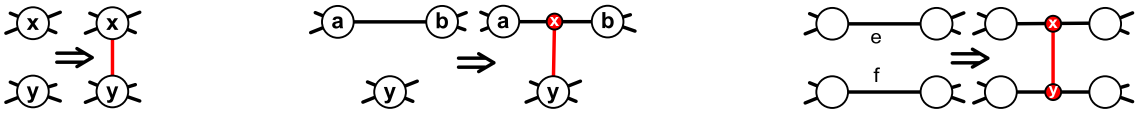
The Barnette and Grünbaum operations (*BG-operations*) consist of the following operations on a 3-connected graph (see Figures 1(a)-1(c)).

(a) add an edge xy (possibly a parallel edge)

(b) subdivide an edge ab by a node x and add the edge xy for a node $y \notin \{a, b\}$

(c) subdivide two distinct, non-parallel edges by nodes x and y, respectively, and add the edge xy

In all three cases, let xy be the edge that was *added* by the BG-operation.



(a) parallel edges allowed

(b) y, a, b distinct

(c) $e \neq f$, e and f not parallel

Figure 1: The three operations of Barnette and Grünbaum.

Theorem 2.4. (Barnette and Grünbaum [2], Tutte [14]) *A graph G is 3-connected if and only if G can be constructed from the K_4 using BG-operations.*

Theorem 2.4 was proven in this notation by Barnette and Grünbaum [2], but implicitly described in a theorem about *nodal connectivity* by Tutte [14, Theorem 12.65]. If not stated otherwise, every construction sequence uses only BG-operations. Let a BG-operation be *basic*, if it does not create parallel edges and let a construction sequence be *basic*, if it only uses basic BG-operations.

Like in Theorem 2.3, we want the inverse of a BG-operation. Let *removing* the edge $e = xy$ of a graph be the operation of deleting e followed by smoothing x and y. An edge $e = xy$ in G is called *removable*, if removing e yields a 3-connected graph. We show that removing a removable edge $e = xy$ with $|N(x)| \geq 3$, $|N(y)| \geq 3$ and $|N(x) \cup N(y)| \geq 5$ is exactly the inverse of a BG-operation.

Theorem 2.5. *The following statements are equivalent:*

$$\begin{aligned} &\textit{A simple graph } G \textit{ is } 3\textit{-connected} && (2.3)\\ \Leftrightarrow\ &\exists \textit{ sequence of removals from } G \textit{ to } K_4 \textit{ on removable edges } e = xy && \\ &\textit{with } |N(x)| \geq 3, |N(y)| \geq 3 \textit{ and } |N(x) \cup N(y)| \geq 5 && (2.4)\\ \Leftrightarrow\ &\exists \textit{ construction sequence from } K_4 \textit{ to } G \textit{ using BG-operations} && (2.5)\\ \Leftrightarrow\ &\exists \textit{ basic construction sequence from } K_4 \textit{ to } G \textit{ using BG-operations} && (2.6)\end{aligned}$$

Proof. Theorem 2.4 establishes (2.3) $\Leftrightarrow$ (2.5). Moreover, the proof of Theorem 2.4 in [2] implicitly shows that on simple graphs basic operations suffice, thus only the equivalence for (2.4) remains. We first prove (2.6) $\Rightarrow$ (2.4) and then (2.4) $\Rightarrow$ (2.5).

BG-operations operate by definition on 3-connected graphs, this holds in particular for the ones in (2.5). Let G' be the graph obtained by a basic BG-operation in (2.5) that adds the edge $e = xy$. The operation can clearly be undone by removing e in G'. Since BG-operations preserve 3-connectedness with Theorem 2.4, $|N(x)| \geq 3$ and $|N(y)| \geq 3$ hold in G'.

It remains to show that $|N(x) \cup N(y)| \geq 5$ in G'. If $|N(x)| \geq 4$ or $|N(y)| \geq 4$, $|N(x) \cup N(y)| \geq 5$ follows, since x and y are neighbors and no self-loops exist. Thus, let $|N(x)| = |N(y)| = 3$. Having $N(x) \setminus \{y\} \neq N(y) \setminus \{x\}$ yields $|N(x) \cup N(y)| \geq 5$ as well, so let $N(x) \setminus \{y\}$ and $N(y) \setminus \{x\}$ contain the same two nodes a and b. If $|V(G)| > 4$, a or b must be adjacent to a node c that is neither adjacent to x nor y. But then $\{a, b\}$ is a separation pair, contradicting the 3-connectedness of G. On the other hand, $|V(G)| = 4$ is not possible, since that implies the BG-operation to be (a) (since only (b) and (c) create new vertices) and that is no basic operation on the K_4.

We prove (2.4) $\Rightarrow$ (2.5). Let G' be the graph containing a removable edge $e = xy$ that is removed in (2.4). Note that G' can have parallel edges due to previous removals but no self-loops. The removal can be undone by one of the BG-operations. Which one, is dependent on the number i of endnodes of e on which smoothing changed the graph, i. e., the number of endnodes u of e with $|N(u)| = deg(u) = 3$ in G'. If $i = 0$, removing e just deletes e which is inversed by operation (a). For $i = 1$, let x be the node with $|N(x)| = deg(x) = 3$ in G' and f be the edge in which x was smoothed. Then (b) can be applied, because $y \notin f$ (see Figure 8(a)) since otherwise x would have had only 2 neighbors in G', contradicting the assumption $|N(x)| \geq 3$.

If $i = 2$, let f_1 and f_2 be the edges in which x and y were smoothed. Operation (c) can only be applied if f_1 and f_2 are neither identical (see Figure 8(b)) nor parallel. But $f_1 = f_2$ would again contradict $|N(x)| \geq 3$ in G' and f_1 being parallel to f_2 would contradicts

$|N(x) \cup N(y)| \geq 5$ in G, since in that case x and y are only adjacent to each other and the two nodes $f_1 \cap f_2$. ■

We show that Barnette and Grünbaum's characterization is algorithmically at least as powerful as Tutte's by giving a simple linear time transformation. Lemma 2.6 allows us to focus on computing BG-operations only.

Lemma 2.6. *Every construction sequence using BG-operations can be transformed in linear time to Tutte's sequence* (2.1) *of contractions.*

Proof. We transform every BG-operation in reverse order of the construction sequence to 0, 1 or 2 contractions each. Operation (a) yields no contraction while operation (b) yields the contraction of exactly one part of the subdivided edge (either xa or xb in Figure 1). For an operation (c), let $e = ab$ and $f = vw$ be the edges that are subdivided with x and y. Both edges share at most one node; let w.l.o.g. $a = v$ be that node if it exists. We create one contraction for each of the edges xb and yw in arbitrary order. In all cases, contractions inverse BG-operations except for the added edge xy, which is left over. But additional edges do not harm the 3-connectedness of the graph nor subsequent contractions. Thus, we have found a contraction sequence to the K_4 unless the first contraction in the case of an operation (c) yields at some point a graph H that is not 3-connected. But H can be obtained from the graph that results from contracting the second edge by applying one operation (b) and therefore is 3-connected. ■

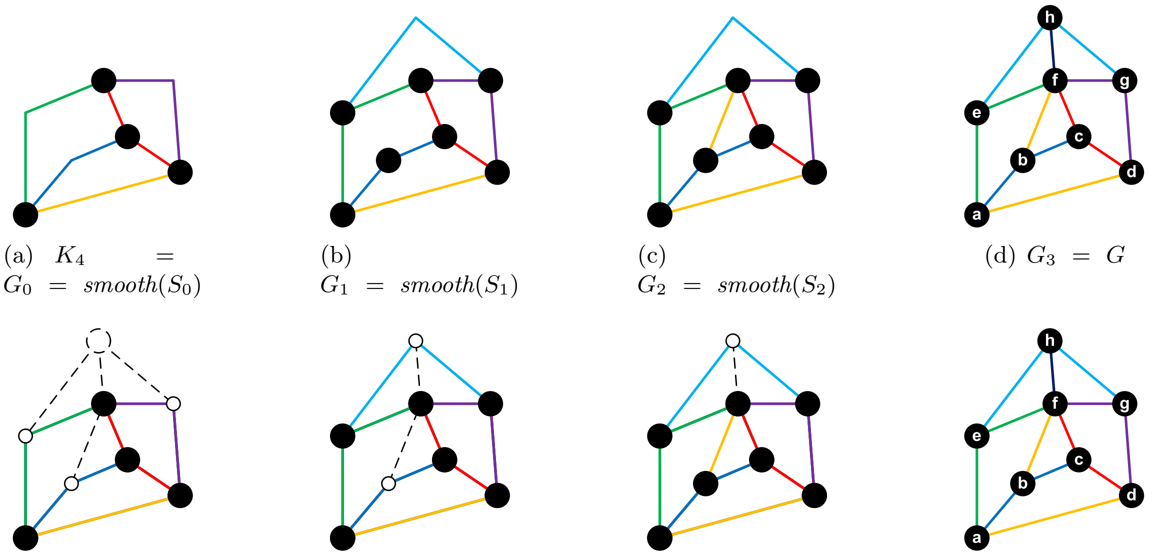
2.3. Identifying Intermediate Graphs with Subdivisions in G

Let $K_4 = G_0, G_1, \ldots, G_z = G$ be the 3-connected graphs obtained in a construction sequence Q to a simple 3-connected graph G using the basic BG-operations $C_0, \ldots, C_{z-1}$. We can reverse Q by starting with G and removing the added edges of BG-operations in reverse order. Suppose we would delete the added edge of every C_i instead of removing it and treat emerging paths containing interior nodes of degree 2 as (topological) edges in G_i (see Figure 2). Then iteratively paths are deleted instead of edges being removed and we obtain the sequence of subdivisions $G = S_z, \ldots, S_0$ in G with S_0 being a subdivision of the K_4. This leads to the following observation.

Lemma 2.7 (Observation)**.** *Let Q be a construction sequence from a graph G_0 to G using BG-operations. Then G contains a subdivision of G_0 that is specified by Q.*

In particular, Observation 2.7 yields with Theorem 2.4 that every 3-connected graph contains a subdivision of the K_4 (Theorem of J. Isbell [2]). Each graph G_i in our construction sequence can be identified with the unique subdivision S_i contained in G. Conversely, $G_i = \mathit{smooth}(S_i)$ for all $0 \leq i \leq z$, since smoothing a graph is exactly the inverse operation of subdividing a graph without nodes of degree two. The nodes x in S_i with $\mathit{deg}(x) \geq 3$ are called *real* nodes, because they correspond to nodes in G_i. Real nodes have at least 3 neighbors in G_i, because G_i is 3-connected.

Note that in non-basic construction sequences $\mathit{smooth}(S_i)$ can have parallel edges, although S_i is always simple. We define the *links* of each S_i to be the unique paths in S_i with only their endnodes being real. The links of S_i partition $E(S_i)$ because S_i is 2-connected, has therefore minimum degree two and is not a cycle. Let two links be *parallel* if they share the same endnodes.



(a) $K_4 = G_0 = smooth(S_0)$ (b) $G_1 = smooth(S_1)$ (c) $G_2 = smooth(S_2)$ (d) $G_3 = G$

(e) S_0 (f) S_1 (g) S_2 (h) $S_3 = G$

Figure 2: The graphs $G_0, \ldots, G_z$ and $S_0, \ldots, S_z$ of a construction sequence of G. On graphs S_i, the dashed edges and nodes are in G but not in S_i and nodes depicted in black are *real* nodes. For example, the path $C_0 = e \rightarrow h \rightarrow g$ is a *BG-path* for S_0, yielding S_1. The *links* of S_1 are the paths C_0, $a \rightarrow b \rightarrow c$ and the single edges ae, ef, fc, cd, da, fg, gd.

Definition 2.8. A *BG-path for S_i* is a path $P = x \rightarrow y$ in G with the following properties:

(1) $S_i \cap P = \{x, y\}$
(2) x and y are not both contained in a link of S_i except as endnodes
(3) x and y are not inner nodes of links of S_i that are parallel

It is easy to see that every BG-path for S_i corresponds to a BG-operation on G_i and vice versa. We will exploit this duality in the next section.

In general, construction sequences are not bound to start with the K_4. Titov and Kelmans [12, 7] extended Theorem 2.4 by proving the existence of a construction sequence even when starting with arbitrary 3-connected graphs G_0 instead of the K_4, as long as a subdivision of G_0 is contained in G. This is a generalization, since every 3-connected graph contains a subdivision of the K_4 by Observation 2.7.

Theorem 2.9. [7, 12] *Let G_0 be a 3-connected graph. Then a simple graph G is 3-connected and contains a subdivision of G_0 if and only if G can be constructed from G_0 using basic BG-operations.*

3. Prescribing Subdivisions

Both Theorems 2.4 and 2.9 choose a very special subdivision of the K_4 (resp. G_0) on which the construction sequence starts, in fact one having the maximum number of edges in G. The construction sequence is then obtained by adding longest BG-paths. Unfortunately, computing these depends heavily on solving the longest paths problem, which is known to be NP-hard even for 3-connected graphs [4].

This gives rise to the question whether Theorems 2.4 and 2.9 can be strengthened to start at a *prescribed* subdivision $H \subseteq G$ of G_0 instead of an arbitrary one. Note that this is equivalent to the constraint $S_0 = H$. Such a result would provide an efficient computational approach to construction sequences, since it allows us to search the neighborhood of H for BG-paths, yielding a new prescribed subdivision of a 3-connected graph.

However, when restricted to basic operations it is not possible to prescribe H, as the minimal counterexample in Figure 3 shows: Consider the graph G consisting of a $K_4 = H$ depicted in black with an additional node connected to three nodes of the K_4. Then every BG-path for H will create a parallel link, although G is simple. But what if we drop the condition that construction sequences have to be basic? The following theorem shows that at this expense we can indeed start a construction sequence from any prescribed subdivision.

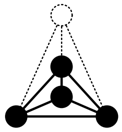


Figure 3: Every possible BG-operation adds a parallel edge.

Theorem 3.1. *Let G be a 3-connected graph and $H \subset G$ with H being a subdivision of a 3-connected graph. Then there is a BG-path for H in G. Moreover, every link of H of length at least 2 contains an inner node on which a BG-path for H starts.*

Proof. We distinguish two cases.

- $H \neq \mathit{smooth}(H)$.
  Then links of length at least 2 exist in H and we pick an arbitrary one of them, say T. Let x be an inner node of T, and let Q be the set of paths in G from x to a node in $V(H) \setminus V(T)$ avoiding the endnodes of T (see Figure 5). By the 3-connectedness of G, the set Q cannot be empty and every path in Q fulfills Definition 2.8.2. There is at least one path $P = x \to y$ in Q with y being not contained in a parallel link of T, because otherwise the endnodes of T would form a separation pair. Let x' be the last node in P that is in T or in a parallel link of T and let y' be the first node after x' that is in $V(H)$. Then $x' \to y'$ has properties 2.8.1 and 2.8.3 and is a BG-path for H.
- $H = \mathit{smooth}(H)$.
  Then H consists only of real nodes and since $H \neq G$, there is a node in $V(G) \setminus V(H)$ or an edge in $E(G) \setminus E(H)$. At first, assume that there is a node $x \in V(G) \setminus V(H)$. Then, by the 2-connectedness of G and Fan Lemma 2.1 we can find a path $P = y_1 \to x \to y_2$ with no other nodes in H than y_1 and y_2. For P the properties 2.8.1-2.8.3 hold, because no link in H can have inner nodes. Let now $V(G) = V(H)$ and e an edge in $E(G) \setminus E(H)$. Then e must be a BG-path for H, since both endnodes are real.

■

In Theorem 3.1, non-basic operations can only occur in the case $H = \mathit{smooth}(H)$ when a path through a node of $V(G) \setminus V(H)$ is chosen. Although we cannot avoid that, it is possible to obtain a basic construction by augmenting the BG-operations with a fourth operation (d).

(d) connect a new node to three distinct nodes

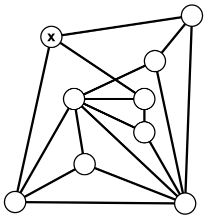


Figure 4: A 3-connected graph having a node x of degree 3 with no incident edge being removable.

Operation (d) preserves 3-connectedness with Lemma 2.2 and is basic, because each new edge ends on the new node. Whenever we encounter a node in $V(G) \setminus V(H)$ in Theorem 3.1, we know by the Fan Lemma 2.1 and the 3-connectedness of G that there are three internally node-disjoint paths to real nodes in H with all inner nodes being in $V(G) \setminus V(H)$. Adding these paths to H is called an *expand* operation and corresponds to operation (d) in the smoothed graph. This gives the following result.

Theorem 3.2. *Let G be a simple graph and let H be a subdivision of a 3-connected graph. Then*

$$\begin{aligned}
&G \text{ is 3-connected and } H \subseteq G \\
\Leftrightarrow\ &\delta(G) \geq 3 \text{ and } \exists \text{ construction sequence from } H \text{ to } G \text{ using BG-paths} \qquad (3.1)\\
\Leftrightarrow\ &\delta(G) \geq 3 \text{ and } \exists \text{ basic construction sequence from } H \text{ to } G \text{ using BG-paths} \\
&\text{and the expand operation} \qquad (3.2)
\end{aligned}$$

Proof. Let G be 3-connected and $H \subseteq G$. Then $\delta(G) \geq 3$ holds and if $H = G$, the desired construction sequences are empty and exist. If $H \subset G$, we can apply Theorem 3.1 iteratively with or without the additional expand operation and the construction sequences exist as well. For the sufficiency part, both construction sequences imply $H \subseteq G$, since only paths are added to construct G. Additionally, G must be 3-connected, as adding BG-paths to each S_i preserves S_{i+1} to be a subdivision of a 3-connected graph with Theorem 2.4, and $\delta(G) \geq 3$ ensures that the last subdivision G of a 3-connected graph is 3-connected itself. ■

4. Representations

A straight-forward algorithm to compute Barnette and Grünbaum's construction sequence of a 3-connected graph is to search iteratively for removable edges. But in contrast to the algorithm in Section 2.1 that computes contractible edges, this approach only leads to an $O(n^3)$ algorithm. The reason for the additional factor of n is that not all nodes with degree 3 must have an incident removable edge (see Figure 4 for a counterexample on 9 nodes) and we have to try every edge in the worst case. Computing BG-paths instead of BG-operations allows us to obtain better running times, but first we need to know how exactly construction sequences can be represented.

An obvious representation of a construction sequence Q would be to store the graph $G_0 = \mathit{smooth}(H)$ and in addition every BG-operation, which gives the sequence $G_0, \ldots, G_z = G$. Unfortunately, the graphs G_i are not necessarily subgraphs of G_{i+1}, so we have to take care of relabeled edges when specifying each operation.

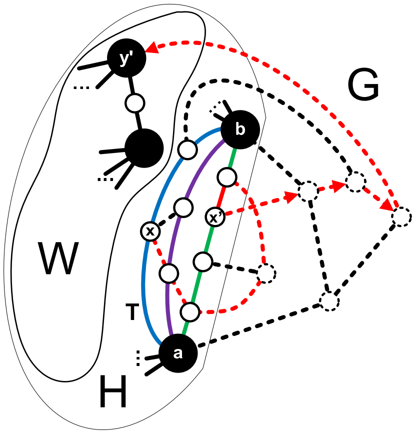


Figure 5: The case $H \neq \mathit{smooth}(H)$. Dashed edges are in $E(G) \setminus E(H)$, arrows depict the BG-path $x' \to y'$.

Whenever an edge e is subdivided as part of an operation (b) or (c), we specify it by its index in G_i followed by assigning new indices for the new degree-two node and one of the two new separated edge parts in G_{i+1}. The other edge part keeps the index of e.

Similarly, on operations (a) and (b), real endnodes of the added edge are specified by their indices in G_i. We assign a new index for the added edge in G_{i+1}, too. Finally, we have to impose the constraint that G_z is not just isomorphic but identical to G, meaning that nodes and edges of G_z and G are labeled by exactly the same indices, since otherwise we would have to solve the graph isomorphism problem to check that Q really constructs G.

On the other hand, the identification of G_i with a subgraph in G allows us to represent Q without indexing issues: We just store $S_0 \subset G$ and the BG-paths $C_0, \ldots, C_{z-1}$. Hence, we can represent each construction sequence Q of G in the following two ways.

- *Edge representation*: Represent Q by G_0 and a sequence of BG-operations, along with specifying new and old indices for each operation, such that G_z and G are labeled the same.
- *Path representation*: Represent Q by S_0 and BG-paths $C_0, \ldots, C_{z-1}$.

Both representations refer to the same sequence of graphs $G_0, \ldots, G_z$ and are of size $\theta(m)$, assuming the uniform cost model. The next lemma states that it does not matter which of the two representations we compute.

Lemma 4.1. *The edge and path representations of a construction sequence Q can be transformed into each other in $O(m)$ time. Moreover, the representation computed is a unique representation of Q.*

Proof. Omitted. ■

5. Certifying and Testing 3-Connectedness in $O(n^2)$

We use construction sequences in the path representation as a certificate for the 3-connectedness of graphs. This leads to a new, certifying method for testing graphs on being

3-connected. The total running time of this method is $O(n^2)$, however this is dominated by the time needed for finding the construction sequence and every improvement made there will automatically result in a faster 3-connectedness test. The input graph is a multigraph and does not have to be biconnected nor connected. We follow the steps:

- Apply preprocessing of Nagamochi and Ibaraki to the graph and get G in $O(n+m)$ (This improves the total running time by decreasing the number of edges to $O(n)$.)
- Try to compute a K_4-subdivision S_0 in G and prescribe it in $O(n)$
  - Failure: Return a separation pair
- Try to compute a construction sequence from S_0 to G in $O(n^2)$
  - Success: Return the construction sequence
  - Failure: Return a separation pair

The preprocessing step preserves the graph to be 3-connected or to be not 3-connected. We first describe how to find a K_4-subdivision by one Depth First Search (DFS), which as a byproduct eliminates self-loops and parallel edges and sorts out graphs that are not connected or have nodes with degree at most 2. Let a (resp. b) be the node in the DFS-tree T that is visited first (resp. second). If G is 3-connected, then a and b have exactly one child, otherwise they form a separation pair. We choose two arbitrary neighbors c and d of a that are different from b (see Figure 6). W.l.o.g., let d be visited later by the DFS than c. Let $i \neq b$ the least common ancestor of c and d in T. As $d \neq i$ must hold, let j be the child of i that is contained in the path $i \to d$ in T.

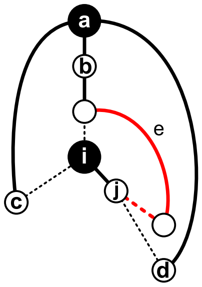


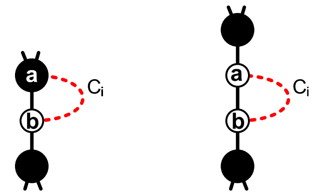
Figure 6: Finding a K_4-subdivision. Dashed edges can be (empty) paths, arcs depict backedges.

If G is 3-connected, we can find a backedge e that starts on a node z in the subtree rooted at j and ends on an inner node z' of $a \to i$ in time $O(n)$. If e does not exist, a and i form a separation pair, otherwise we have found a K_4-subdivision with real nodes a, i, z and z'. The paths connecting this real nodes in T together with the three visited backedges constitute the 6 paths of the K_4-subdivision.

Once the K_4-subdivision S_0 is found, we follow the lines of Theorem 3.1 and try to construct the path representation $C_0, \dots, C_{z-1}$. If favored, this can be transformed to an edge representation in $O(m)$ later. We assign an index for every link and store it on each of the inner nodes of that link. Moreover, we maintain pointers for each link to its endnodes.

In case $H \neq \mathit{smooth}(H)$ of Theorem 3.1 we pick an arbitrary node x of degree two. Let $T = a \to b$ be the link that contains x and let W be the set of nodes $V(H) \setminus V(T)$ minus all nodes in parallel links of T (see Figure 5). We compute the path $P = x \to y'$ by temporarily deleting a and b and performing a DFS on x that stops on the first node $y' \in W$. We can check whether a node lies in a parallel link of T in constant time by comparing the endnodes of its containing link with a and b. Thus, the subpath $x' \to y'$ with x' being the last node contained in T or in a parallel link of T is a BG-path and can be found efficiently. The links and their indices can be updated in $O(n)$.

Similarly, in case $H = \mathit{smooth}(H)$ we delete temporarily all edges in $E(H)$ and start a DFS on a node $x \in V(H)$ that has an incident edge in the remaining graph. The traversal is stopped on the first node $y \in V(H) \setminus \{x\}$. The path $x \to y$ is then the desired BG-path



(a) Either a or b has degree 2.

(b) Both, a and b, have degree 2.

Figure 8: Cases where 2.8.2 fails when $a \in N(b)$.

and we conclude that for 3-connected graphs the construction sequence can be found in time $O(n^2)$.

Otherwise, G is not 3-connected and no construction sequence can exist with Theorem 3.2. In that case a DFS starting at node x fails to find a new BG-path for some subdivision $H \subset G$. If $H \neq \mathit{smooth}(H)$, the endnodes of the link that contains x must form a separation pair. Otherwise, $H = \mathit{smooth}(H)$ and x must be a cut vertex. Thus, if G is not 3-connected, the algorithm returns always a separation pair or cut vertex.

If G is simple, the construction sequence can be transformed to the basic construction sequence (3.2) with the following Lemma.

Lemma 5.1. *For simple graphs G, the construction sequences* (3.1) *and* (3.2) *can be transformed into each other in $O(m)$.*

Proof. Omitted. ■

Theorem 5.2. *The construction sequences* (3.1) *and* (3.2) *can be computed in $O(n^2)$ and establish a certifying* 3*-connectedness test with the same running time.*

5.1. Verifying the Construction Sequence

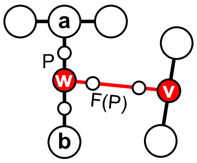


Figure 7: No expand operation can be formed.

It is essential for a certificate that it can be easily validated. We could do this by transforming the path representation to the edge representation using Lemma 4.1 and checking the validity of the BG-operations by comparing indices, but there is a more direct way. First, it can be checked in linear time that all BG-paths $C_0, \ldots, C_{z-1}$ are paths in G and that these paths partition $E(G) \setminus E(S_0)$. We try to remove the BG-paths $C_{z-1}, \ldots, C_0$ from G in that order (i. e., we delete the paths followed by smoothing its endnodes). If the certificate is valid, this is well defined as all removed BG-paths are then edges. On the other hand we can detect longer BG-paths $|C_i| \geq 2$ before their removal, in which case the certificate is not valid, since then the inner nodes of C_i are not attached to BG-paths C_j, $j > i$.

We verify that every removed $C_i = ab$ corresponds to a BG-operation by using Definition 2.8 of BG-paths, and start with checking that a and b lie in our current subgraph for condition 2.8.1.

Conditions 2.8.2 and 2.8.3 can now be checked in constant time: Consider the situation immediately after the deletion of ab, but before smoothing a and b. Then all links in our subgraph are single edges, except possibly the ones containing a and b as inner nodes.

Therefore, 2.8.2 is not met for C_i if a is a neighbor of b and at least one of the nodes a and b has degree two (see Figures 8 for possible configurations). Condition 2.8.3 is not met if $N(a) = N(b)$ and both a and b have degree two. Both conditions can be easily checked in constant time. Note that encountering proper BG-paths $C_{z-1}, \ldots, C_i$ does not necessarily imply that the current subgraph is 3-connected, since false BG-paths C_j, $j < i$, can exist.

It remains to validate that the graph after removing all BG-paths is the K_4. This can done in constant time by checking it on being simple and having exactly 4 nodes of degree three.

Theorem 5.3. *The construction sequences* (2.4)-(2.6) *and* (3.1)-(3.2) *can be checked on validity in time linearly dependent on their length.*

References

[1] S. Albroscheit. Ein Algorithmus zur Konstruktion gegebener 3-zusammenhängender Graphen. Diploma thesis, FU Berlin, 2006.

[2] D. W. Barnette and B. Grünbaum. On Steinitz's theorem concerning convex 3-polytopes and on some properties of 3-connected graphs. *Many Facets of Graph Theory, Lecture Notes in Mathematics*, 110:27–40, 1969.

[3] M. Blum and S. Kannan. Designing programs that check their work. In *STOC '89*, pages 86–97, New York, 1989.

[4] M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar hamiltonian circuit problem is NP-complete. *Siam J. Comp.*, 5(4):704–714, 1976.

[5] R. Halin. Zur Theorie der n-fach zusammenhängenden Graphen. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 33(3):133–164, 1969.

[6] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

[7] A. K. Kelmans. Graph expansion and reduction. *Algebraic methods in graph theory, Szeged, Hungary*, 1:317–343, 1978.

[8] K. Menger. Zur allgemeinen Kurventheorie. *Fund. Math.*, 10:96–115, 1927.

[9] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(1-6):583–596, 1992.

[10] C. Thomassen. Kuratowski's theorem. *Journal of Graph Theory*, 5(3):225–241, 1981.

[11] C. Thomassen. Reflections on graph theory. *Journal of Graph Theory*, 10(3):309–324, 2006.

[12] V. K. Titov. *A constructive description of some classes of graphs*. PhD thesis, Moscow, 1975.

[13] W. T. Tutte. A theory of 3-connected graphs. *Indag. Math.*, 23:441–455, 1961.

[14] W. T. Tutte. Connectivity in graphs. In *Mathematical Expositions*, volume 15. University of Toronto Press, 1966.

[15] K.-P. Vo. Finding triconnected components of graphs. *Linear and Multilinear Algebra*, 13:143–165, 1983.

[16] K.-P. Vo. Segment graphs, depth-first cycle bases, 3-connectivity, and planarity of graphs. *Linear and Multilinear Algebra*, 13:119–141, 1983.

[17] D. B. West. *Introduction to Graph Theory.* Prentice Hall, 2001.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 645-656
www.stacs-conf.org

NAMED MODELS IN COALGEBRAIC HYBRID LOGIC

LUTZ SCHRÖDER [1] AND DIRK PATTINSON [2]

[1] DFKI Bremen and Department of Computer Science, Universität Bremen
E-mail address: Lutz.Schroeder@dfki.de

[2] Department of Computing, Imperial College London
E-mail address: dirk@doc.ic.ac.uk

ABSTRACT. Hybrid logic extends modal logic with support for reasoning about individual states, designated by so-called nominals. We study hybrid logic in the broad context of coalgebraic semantics, where Kripke frames are replaced with coalgebras for a given functor, thus covering a wide range of reasoning principles including, e.g., probabilistic, graded, default, or coalitional operators. Specifically, we establish generic criteria for a given coalgebraic hybrid logic to admit named canonical models, with ensuing completeness proofs for pure extensions on the one hand, and for an extended hybrid language with local binding on the other. We instantiate our framework with a number of examples. Notably, we prove completeness of graded hybrid logic with local binding.

Introduction

Modal logics have traditionally played a central role in Computer Science, appearing, e.g., in the guise of temporal logics, program logics such as PDL, epistemic logics, and later as description logics. The development of modal logics has seen extensions along (at least) two axes: the enhancement of the expressive power of basic (relational) modal logic on the one hand, and the continual extension, beyond the purely relational realm, of the class of structures described using modal logics on the other hand. Hybrid logic falls into the first category, extending modal logic with the ability to reason about individual states in models. This feature, originally suggested by Prior and first studied in the context of tense logics and PDL (see [5] for references), is of particular relevance in knowledge representation languages and as such has found its way into modern description logics, where it is denoted by the letter $\mathcal{O}$ in the standard naming scheme [2].

Extensions along the second axis – semantics beyond Kripke structures and neighbourhood models – include various probabilistic modal logics, interpreted over probabilistic transition systems, graded modal logic over multigraphs [8], conditional logics over selection function frames [6], and coalition logic [17], interpreted over so-called game frames. As a unifying semantic bracket covering all these logics and many further ones, coalgebraic modal logic has emerged ([7] gives a survey). The scope of coalgebraic modal logic has recently been expanded to encompass nominals; we refer to the arising class of logics as *coalgebraic hybrid logics*. Existing results include a finite

1998 ACM Subject Classification: F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic — modal logic; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods — modal logic, representation languages.

Key words and phrases: Logic in computer science, semantics, deduction, modal logic, coalgebra.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2492

model result, an internalized tableaux calculus, and generic $PSPACE$ upper bounds, but are so far limited to logics that exclude frame conditions and local binding [14]. What is missing from this picture technically is a theory of *named canonical models* [5]. Named canonical models yield not only strong completeness of the basic hybrid logic, but also completeness of *pure extensions*, defined by axioms that do not contain propositional variables (but may contain nominals; e.g. in Kripke semantics, the pure axiom $\Diamond\Diamond i \to \Diamond i$, with i a *nominal*, defines transitive frames). Moreover, named canonical models establish completeness for an extended hybrid language with a local binding operator $\downarrow x.\, \phi(x)$, read as "the current state x satisfies $\phi(x)$". Both pure extensions and the language with $\downarrow$ (not addressed in [14]) are, in general, undecidable [1] (it should be noted, however, that fragments of the language with $\downarrow$ over Kripke frames are decidable and as such play a role, e.g., in conjunctive query answering in description logic [11]). As a consequence, completeness of pure extensions and local binding is the best we can hope for – it establishes recursive enumerability of the set of valid formulas, and it enables automated reasoning, if not decision procedures.

Specifically, we establish two separate criteria for the existence of named models. Although these criteria are (in all likelihood necessarily) less widely applicable than some previous coalgebraic results including those of [14], the generic results allow us to establish new completeness results for a wide variety of logics; in particular, we prove strong completeness of graded hybrid logic, and ultimately an extension of the description logic $\mathcal{SHOQ}$, with the $\downarrow$ binder over a wide variety of frame classes.

1. Coalgebraic Hybrid Logic

To make our treatment parametric in the syntax, we fix a modal similarity type Λ consisting of modal operators with associated arities throughout. For given countably infinite and disjoint sets P of propositional variables and N of nominals, the set $\mathcal{F}(\Lambda)$ of *hybrid* Λ*-formulas* is given by the grammar

$$\mathcal{F}(\Lambda) \ni \phi, \psi ::= p \mid i \mid \phi \wedge \psi \mid \neg\phi \mid \heartsuit(\phi_1, \dots, \phi_n) \mid @_i\phi$$

where $p \in \mathsf{P}$, $i \in \mathsf{N}$ and $\heartsuit \in \Lambda$ is an n-ary modal operator. (Alternatively, we could regard *propositional variables* as nullary modal operators, thus avoiding their explicit mention altogether. We keep them explicit here, following standard practice in modal logic, as we have to deal with valuations anyway due to the presence of nominals.) We use the standard definitions for the other propositional connectives $\to, \leftrightarrow, \vee$. The set of nominals occurring in a formula ϕ is denoted by $\mathsf{N}(\phi)$, similarly for sets of formulas. A formula of the form $@_i\phi$ is called an $@$*-formula*. Semantically, nominals i denote individual states in a model, and $@_i\phi$ stipulates that ϕ holds at state i.

To reflect parametricity also semantically, we equip hybrid logics with a *coalgebraic semantics* extending the standard coalgebraic semantics of modal logics [16]: we fix throughout a Λ*-structure* consisting of an endofunctor $T : \mathsf{Set} \to \mathsf{Set}$ on the category of sets, together with an assignment of an n*-ary predicate lifting* $[\![\heartsuit]\!]$ to every n-ary modal operator $\heartsuit \in \Lambda$, i.e. a set-indexed family of mappings $([\![\heartsuit]\!]_X : \mathcal{P}(X)^n \to \mathcal{P}(TX))_{X \in \mathsf{Set}}$ that satisfies

$$[\![\heartsuit]\!]_X \circ (f^{-1})^n = (Tf)^{-1} \circ [\![\heartsuit]\!]_Y$$

for all $f : X \to Y$. In categorical terms, $[\![\heartsuit]\!]$ is a natural transformation $\mathcal{Q}^n \to \mathcal{Q} \circ T^{op}$ where $\mathcal{Q} : \mathsf{Set}^{op} \to \mathsf{Set}$ is the contravariant powerset functor.

In this setting, T-coalgebras play the roles of *frames*. A T*-coalgebra* is a pair (C, γ) where C is a set of *states* and $\gamma : C \to TC$ is the *transition function*. When γ is clear from the context, we refer to (C, γ) just as C. A *(hybrid)* T*-model* $M = (C, \gamma, V)$ consists of a T-coalgebra (C, γ) together with a *hybrid valuation* V, i.e. a map $\mathsf{P} \cup \mathsf{N} \to \mathcal{P}(C)$ that assigns singleton sets to all nominals $i \in \mathsf{N}$. We say that M is *based* on the frame (C, γ). The singleton set $V(i)$ is tacitly identified with its unique element.

The semantics of $\mathcal{F}(\Lambda)$ is a satisfaction relation $\models$ between states $c \in C$ in hybrid T-models $M = (C, \gamma, V)$ and formulas $\phi \in \mathcal{F}(\Lambda)$, inductively defined as follows. For $x \in \mathsf{N} \cup \mathsf{P}$ and $i \in \mathsf{N}$,

$$M, c \models x \text{ iff } c \in V(x) \qquad \text{and} \qquad M, c \models @_i\phi \text{ iff } M, V(i) \models \phi.$$

Modal operators are interpreted using their associated predicate liftings, that is,

$$M, c \models \heartsuit(\phi_1, \dots, \phi_n) \iff \gamma(c) \in [\![\heartsuit]\!]_C([\![\phi_1]\!]_M, \dots, [\![\phi_n]\!]_M)$$

where $\heartsuit \in \Lambda$ is n-ary and $[\![\phi]\!]_M = \{c \in C \mid M, c \models \phi\}$ denotes the truth-set of ϕ relative to M. We write $M \models \phi$ if $M, c \models \phi$ for all $c \in C$. For a set $\Phi \subseteq \mathcal{F}(\Lambda)$ of formulas, we write $M, c \models \Phi$ if $M, c \models \phi$ for all $\phi \in \Phi$, and $M \models \Phi$ if $M \models \phi$ for all $\phi \in \Phi$. We say that Φ is *satisfiable* in a model M if there exists a state c in M such that $M, c \models \Phi$. If $\mathcal{A} \subseteq \mathcal{F}(\Lambda)$ is a set of axioms, also referred to as *frame conditions*, a frame (C, γ) is an *$\mathcal{A}$-frame* if $(C, \gamma, V) \models \phi$ for all hybrid valuations V and all $\phi \in \mathcal{A}$, and a model is an *$\mathcal{A}$-model* if it is based on an $\mathcal{A}$-frame. A frame condition is *pure* if it does not contain any propositional variables (it may however contain nominals). We recall notation from earlier work:

Notation 1. As usual, application of substitutions $\sigma : \mathsf{P} \to \mathcal{F}(\Lambda)$ to formulas ϕ is denoted $\phi\sigma$. For a set Σ of formulas and a set O of operators, we write $O\Sigma$ or $O(\Sigma)$ for the set of formulas arising by prefixing elements of Σ with an operator from O; e.g. $\Lambda(\Sigma) = \{\heartsuit(\phi_1, \dots, \phi_n) \mid \heartsuit \in \Lambda\ n\text{-ary}, \phi_1, \dots, \phi_n \in \Sigma\}$ and $@\Sigma := \{@_i \mid i \in \mathsf{N}\}(\Sigma) = \{@_i\phi \mid i \in \mathsf{N}, \phi \in \Sigma\}$. Moreover, $\mathsf{Prop}(Z)$ denotes the set of propositional combinations of elements of some set Z. For $\phi \in \mathsf{Prop}(Z)$, we write $X, \tau \models \phi$ if ϕ evaluates to $\top$ in the boolean algebra $\mathcal{P}(X)$ under a valuation $\tau : Z \to \mathcal{P}(X)$. For $\psi \in \mathsf{Prop}(\Lambda(Z))$, the interpretation $[\![\psi]\!]_{TX,\tau}$ of ψ in the boolean algebra $\mathcal{P}(TX)$ under τ is the inductive extension of the assignment $[\![\heartsuit(p_1, \dots, p_n)]\!]_{TX,\tau} = [\![\heartsuit]\!]_X(\tau(p_1), \dots, \tau(p_n))$. We write $TX, \tau \models \psi$ if $[\![\psi]\!]_{TX,\tau} = TX$, and $t \models_{TX,\tau} \psi$ if $t \in [\![\psi]\!]_{TX,\tau}$. A set of formulas $\Xi \subseteq \mathsf{Prop}(\Lambda(Z))$ is *one-step satisfiable* w.r.t. τ if $\bigcap_{\phi \in \Xi} [\![\phi]\!]_{TX,\tau} \neq \emptyset$. We occasionally apply this notation to sets $Z \subseteq \mathcal{P}(X)$ with τ being just inclusion, in which case mention of τ is suppressed.

In the sequel, we will be interested in both *local* and *global* semantic consequence, where local consequence refers to satisfaction in a single state and global consequence to satisfaction in entire models. In fact, we consider local reasoning under global assumptions: given a set $\Phi \subseteq \mathcal{F}(\Lambda)$ of global assumptions (a *TBox* in description logic terminology) and a class $\mathcal{C}$ of models, we say that *ϕ is a local consequence of Ψ under global assumptions Φ for $\mathcal{C}$-models*, in symbols $\Phi; \Psi \models^{\mathcal{C}} \phi$, if for all $M \in \mathcal{C}$ such that $M \models \Phi$, $M, c \models \phi$ whenever $M, c \models \Psi$ (here, both Φ and Ψ are sets of arbitrary formulas, in particular not subject to any restrictions on the nesting depth of modal operators). The standard notions of local and global consequence are regained from this general definition by taking Φ or Ψ to be empty, respectively.

The distinguishing feature of the coalgebraic approach to hybrid and modal logics is the parametricity in both the logical language and the notion of frame: concrete instantiations of the general framework, in other words a choice of modal operators Λ and a Λ-structure T, capture the syntax and semantics of a wide range of modal logics, as witnessed by the following examples.

Examples 1.1. 1. The hybrid version of the modal logic K, *hybrid K* for short, has a single unary modal operator $\Box$, interpreted over the structure consisting of the powerset functor $\mathcal{P}$ (which takes a set X to its powerset $\mathcal{P}(X)$) and the predicate lifting $[\![\Box]\!]_X(A) = \{B \in \mathcal{P}(X) \mid B \subseteq A\}$. It is clear that $\mathcal{P}$-coalgebras $(C, \gamma : C \to \mathcal{P}(C))$ are in 1-1 correspondence with Kripke frames, and that the coalgebraic definition of satisfaction specializes to the usual semantics of the box operator.

2. *Graded hybrid logic* has modal operators $\Diamond_k$ 'in more than k successors, it holds that'. It is interpreted over the functor $\mathcal{B}$ that takes a set X to the set $\mathcal{B}(X) = X \to \mathbb{N} \cup \{\infty\}$ of multisets over X by $[\![\Diamond_k]\!]_X(A) = \{B \in \mathcal{B}(X) \mid \sum_{x \in A} B(x) > k\}$. This captures the semantics of graded modalities over *multigraphs* [8], which are precisely the $\mathcal{B}$-coalgebras. A more general set of operators is that of *Presburger logic* [9], which admits integer linear inequalities $\sum a_i \cdot \#(\phi_i) \geq k$ among formulas. Unlike in the purely modal case [19], hybrid multigraph semantics visibly differs from the more standard Kripke semantics of graded modalities, as the latter validates all formulas $\neg\Diamond_1 i$, $i \in \mathsf{N}$. However, both semantics agree if we additionally stipulate $\neg\Diamond_1 i$ as a global (pure) axiom. Thus, our completeness results for multigraph semantics derived below do transfer to Kripke semantics. In particular they apply to many description logics, which commonly feature both nominals and graded modal operators in the guise of *qualified number restrictions*.

3. *Hybrid CK*, the hybrid extension of the basic conditional logic CK, has a single binary modal operator $\Rightarrow$, written in infix notation. Hybrid CK is interpreted over the functor Cf that maps a set X to the set $\mathcal{P}(X) \to \mathcal{P}(X)$, whose coalgebras are selection function models [6], by putting $[\![\Rightarrow]\!]_X(A, B) = \{f : \mathcal{P}(X) \to \mathcal{P}(X) \mid f(A) \subseteq B\}$.

4. *Classical hybrid logic* (the hybrid version of the logic E of neighbourhood frames, referred to as (the minimal) classical modal logic in [6]) has a single, unary modal operator $\Box$ and is interpreted over *neighbourhood frames*, that is, coalgebras for the functor $NX = \mathcal{P}(\mathcal{P}(X))$ (more precisely, the double contravariant powerset functor). The semantics of classical modal logic is defined by the lifting $[\![\Box]\!]_X(A) = \{S \in NX \mid A \in S\}$. *Monotone hybrid logic* has the same similarity type, but is interpreted over upwards closed neighbourhood frames, or coalgebras for the functor $MX = \{S \in NX \mid S$ upwards closed$\}$ where upwards closure refers to subset inclusion.

5. The syntax of coalition logic over a set N of agents is given by the similarity type $\{[C] \mid C \subseteq N\}$, and the operator $[C]$ reads as "coalition C has a joint strategy to enforce ...". The formulas of (hybrid) coalition logic are interpreted over game frames, i.e., coalgebras for the functor

$$\mathsf{G}(X) = \{(f, (S_i)_{i \in N}) \mid \textstyle\prod_{i \in N} S_i \neq \emptyset, f : \prod_{i \in N} S_i \to X\}$$

(a class-valued functor, technically speaking, which however does not cause problems). The semantics arises via the liftings

$$[\![[C]]\!]_X(A) = \{(f, (S_i)_{i \in N}) \in \mathsf{G}(X) \mid \exists (s_i)_{i \in C} \forall (s_i)_{i \in N \setminus C} (f((s_i)_{i \in N}) \in A\}.$$

We proceed to present a Hilbert-style proof system for coalgebraic hybrid logics, which we prove to be sound and strongly complete. This requires that the logic at hand satisfies certain coherence conditions between the axiomatization and the semantics — in fact the *same* conditions as in the purely modal case, which are easily verified *local* properties that can be verified without reference to T-models and are already known to hold for a large variety of logics [16, 19].

Proof systems for coalgebraic logics are most conveniently described in terms of one-step rules, as follows.

Definition 1.2. A *one-step rule* over Λ is a rule ϕ/ψ where $\phi \in \mathsf{Prop}(\mathsf{P})$ and $\psi \in \mathsf{Prop}(\Lambda(\mathsf{P}))$ (in fact, ψ may be restricted to be a disjunctive clause, which however is not relevant here). The rule ϕ/ψ is *one-step sound* if $TX, \tau \models \psi$ whenever $X, \tau \models \phi$ for a valuation $\tau : \mathsf{P} \to \mathcal{P}(X)$. Given a set $\mathcal{R}$ of one-step rules and a valuation $\tau : \mathsf{P} \to \mathcal{P}(X)$, a set $\Xi \subseteq \mathsf{Prop}(\Lambda(\mathsf{P}))$ is *one-step consistent* [20] if the set $\Xi \cup \{\psi\sigma \mid \sigma : \mathsf{P} \to \mathsf{Prop}(\mathsf{P}); \phi/\psi \in \mathcal{R}; X, \tau \models \phi\sigma\}$ is propositionally consistent.

One-step sound rules are sound, and we will assume one-step soundness tacitly in the sequel. Completeness hinges on variants of the notion of one-step completeness [19], which we define further below. As the notion of one-step rule does not involve hybrid features, suitable rule sets can just be inherited from the corresponding modal systems; for graded logics, conditional logics, and many others, such rule sets are found, e.g., in [22, 21]. We recall that the one-step complete rule set for (hybrid) K consists of the rules

$$\frac{a}{\Box a} \qquad \frac{a \wedge b \to c}{\Box a \wedge \Box b \to \Box c}\,.$$

A set $\mathcal{R}$ of one-step rules now gives rise to a Hilbert system $\mathcal{LR}$ by adjoining propositional tautologies and the hybrid axioms, and closing under modus ponens, rule application, and @-necessitation. Formally, we write $\Phi \vdash_{\mathcal{LR}} \phi$ for a set Φ of formulas, the *global assumptions* (or the *TBox*), and a formula ϕ if ϕ is contained in the smallest set that

- contains Φ and all instances of propositional tautologies
- contains all instances of @-introduction $i \wedge \phi \to @_i\phi$ and make-or-break

$$\text{(mob)} \quad @_i p \to (\heartsuit(q_1, \dots, q_n) \leftrightarrow \heartsuit(@_i p \wedge q_1, \dots, @_i p \wedge q_n))$$

  together with all instances of the axioms $\neg @_i \bot$, $\neg @_i \phi \leftrightarrow @_i \neg\phi$, $@_i(\phi \wedge \psi) \leftrightarrow (@_i\phi \wedge @_i\psi))$, $@_i i$, $@_i j \leftrightarrow @_j i$, $@_i k \wedge @_j p \to @_i p$; and
- is closed under instances of @-generalization $p/@_i p$, instances of rules in $\mathcal{R}$, and modus ponens.

The second group of axioms ensures that $i \sim j :\equiv @_i j$ defines an equivalence relation on nominals and that $@_i$ distributes over propositional connectives. The (mob) axiom captures the fact that the truth set of an @-formula is either empty or the whole model; in the case of hybrid K, it is equivalent to the standard back axiom $@_i\phi \to \Box @_i\phi$.

We write $\Phi; \Psi \vdash_{\mathcal{LR}} \phi$ if there are $\psi_1, \dots, \psi_n \in \Psi$ such that $\Phi \vdash_{\mathcal{LR}} \psi_1 \wedge \dots \wedge \psi_n \to \phi$. That is, $\Phi; \Psi \vdash_{\mathcal{LR}} \phi$ if there is a proof of ϕ from global assumptions Φ that additionally assumes Ψ *locally*. As we assume that all one-step rules in $\mathcal{R}$ are one-step sound, soundness for both local and global consequence is immediate: we have $\Phi; \Psi \models^{\mathcal{C}} \phi$ (for $\mathcal{C}$ the class of all models) whenever $\Phi; \Psi \vdash_{\mathcal{LR}} \phi$. In [14], a criterion has been given for $\mathcal{LR}$ to be *weakly complete*, i.e. complete for the case where both the TBox Φ and the set Ψ of local assumptions are empty. Here, we extend this result to combined strong *global* and strong *local* completeness, i.e. to cover both an arbitrary TBox and an arbitrary set of local assumptions, even if $\mathcal{LR}$ is extended with pure frame conditions and local binding.

2. Strong Completeness of Pure Extensions

Pure completeness is a celebrated result in hybrid logic [3, Chapter 7.3]. In a nutshell, adding pure axioms to an already complete proof system for the hybrid extension of the modal logic K (Example 1.1), one retains completeness with respect to the class of frames that satisfy the additional axioms. In contrast to arbitrary modal axioms, pure axioms do not contain propositional variables, and therefore define – in the classical setting of hybrid K – first-order frame conditions. Here, we show that the same theorem is valid for a much larger class of logics, namely all coalgebraic hybrid logics satisfying one of two suitable sets of conditions. For the sake of readability, we restrict the technical development (not the examples) to the case of unary operators from now on until Section 2.2.

Definition 2.1. If $\mathcal{A}$ is a set of pure formulas and $\mathcal{R}$ is a set of one-step rules, we write $\Phi; \Psi \vdash_{\mathcal{LRA}+\mathsf{Name}} \phi$ if there are $\psi_1, \dots, \psi_n \in \Psi$ such that $\psi_1 \wedge \dots \psi_n \to \phi$ is $\mathcal{LR}$-derivable from assumptions in Φ where additionally all substitution instances of axioms in $\mathcal{A}$ and the rule

$$(\mathsf{Name})\frac{i \to \phi}{\phi}(i \notin \mathsf{N}(\phi))$$

may be used in deductions. As before, we write $\Phi \vdash_{\mathcal{LRA}+\mathsf{Name}} \phi$ if $\Phi; \emptyset \vdash_{\mathcal{LRA}+\mathsf{Name}} \phi$.

In the above system, the rule (Name') $@_i\phi/\phi$ $(i \notin \mathsf{N}(\phi)$ and the rule

$$(\mathsf{NameCong})\ \frac{@_j(\phi \leftrightarrow \psi)}{\heartsuit\phi \leftrightarrow \heartsuit\psi}\ (j \notin \mathsf{N}(\phi, \psi))$$

are derivable. The system is clearly sound for both global and local consequence over $\mathcal{A}$-models in the same sense that $\mathcal{LR}$ is sound over T-models.

Definition 2.2. Let $\mathcal{A} \subseteq \mathcal{F}(\Lambda)$ be a set of pure axioms, and let $\Phi \subseteq \mathcal{F}(\Lambda)$ be a TBox. A set $\Psi \subseteq \mathcal{F}(\Lambda)$ is $(\mathcal{LRA} + \mathsf{Name})$-$\Phi$-*inconsistent* if there are $\psi_1, \dots, \psi_n \in \Psi$ such that $\Phi \vdash_{\mathcal{LRA}+\mathsf{Name}} \neg(\psi_1 \wedge \dots \wedge \psi_n)$. Otherwise, Ψ is $(\mathcal{LRA} + \mathsf{Name})$-$\Phi$-*consistent*. A subset of $@\mathcal{F}(\Lambda)$, i.e. a set of $@$-formulas, is called an *ABox* (again borrowing terminology from description logic). A *maximally* $(\mathcal{LRA}+\mathsf{Name})$-$\Phi$-*consistent ABox* is a maximal element K among the $(\mathcal{LRA}+\mathsf{Name})$-$\Phi$-consistent ABoxes, ordered by inclusion. For such a K, we write $S_K = \{K_i \mid i \in \mathsf{N}\}$, where $K_i = \{\phi \in \mathcal{F}(\Lambda) \mid @_i\phi \in K\}$, and put $V_K(i) = \{K_i\} = \{K_j \in S_K \mid i \in K_j\}$.

For the construction of a named model, we now fix a maximally $(\mathcal{LRA}+\mathsf{Name})$-$\Phi$-consistent ABox K. Later, we will take K to be a maximally consistent extension of a given set Φ of formulas, where we may assume, thanks to the rule (Name'), that $\Phi \subseteq @\mathcal{F}(\Lambda)$. We note the following trivial facts:

Lemma 2.3. *We have $\psi\sigma \in K_i$ for all $\psi \in \mathcal{A}$ and all substitutions σ, and moreover $K \cup \Phi \subseteq K_i$.*

Our goal is the construction of named canonical models in the following sense:

Definition 2.4. A *named canonical K-model* is a model (S_K, γ, V_K) such that

$$\gamma(K_i) \in [\![\heartsuit]\!]\hat{\phi} \quad \text{iff} \quad \heartsuit\phi \in K_i$$

for every nominal i, where $\hat{\phi} = \{K_j \in S_K \mid \phi \in K_j\}$.

It is clear that named canonical models are countable, as there are only countably many nominals.

Lemma 2.5 (Truth lemma for named canonical models)**.** *If $M = (S_K, \gamma, V_K)$ is a named canonical K-model and ϕ is a hybrid formula, then for every $K_i \in S_K$,*

$$M, K_i \models \phi \quad \textit{iff} \quad \phi \in K_i.$$

Hence, $M \models \Phi$, and M is an $\mathcal{A}$-model.

The last clause of the truth lemma follows from Lemma 2.3, the crucial point being that satisfaction of all substitution instances of $\mathcal{A}$ implies frame satisfaction of $\mathcal{A}$ because every state in the model is denoted by some nominal. We now establish two criteria for the existence of named canonical models. The first criterion assumes a stronger form of one-step completeness than the second, which instead demands that the modalities are *bounded*.

2.1. Pure Completeness for Strongly One-Step Complete Logics

The construction of named models hinges on the following notion of pastedness, which assures that nominals interact correctly across the whole model. For the rest of the section, we fix a one-step complete rule set $\mathcal{R}$, a set $\mathcal{A}$ of pure axioms, and a set $\Phi \subseteq \mathcal{F}(\Lambda)$ of global assumptions, and we write 'consistent' instead of '$(\mathcal{LRA} + \mathsf{Name})$-$\Phi$–consistent'.

Definition 2.6. An ABox K is 0-*pasted* if whenever $@_j(\phi \leftrightarrow \psi) \in K$ for all nominals j, then $@_i(\heartsuit\phi \leftrightarrow \heartsuit\psi) \in K$ for all nominals i.

It is clear that K can induce a named model only if K is 0-pasted. The construction of pasted ABoxes requires a Henkin-like extension of the logical language by adding new nominals. *Generally, we denote by $\mathcal{F}(\Lambda)^+$ an extended language with countably many new nominals not appearing in $\mathcal{F}(\Lambda)$.* We note the fact (slightly glossed over in the literature) that this extension is conservative:

Lemma 2.7. *If $\Psi \subseteq \mathcal{F}(\Lambda)$ is consistent, then Ψ remains consistent in $\mathcal{F}(\Lambda)^+$.*

Lemma 2.8 (Extended Lindenbaum lemma for 0-Pasted Sets)**.** *If $\Psi \subseteq \mathcal{F}(\Lambda)$ is consistent, then there exists a 0-pasted maximally consistent ABox $K \subseteq @\mathcal{F}(\Lambda)^+$ and a nominal i in $\mathcal{F}(\Lambda)^+$ such that $@_i\Psi \subseteq K$.*

(The proof of the above version of the Lindenbaum lemma uses Lemma 2.7, and exploits the Name' rule to introduce the nominal i.) As we are aiming for strong completeness results, (weak) one-step completeness as employed in weak completeness proofs using *finite* models [14, 19] is no longer adequate. Accordingly, our first criterion assumes a stronger condition:

Definition 2.9. A rule set $\mathcal{R}$ is *strongly one-step complete* if for every set X, every one-step consistent subset of $\mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ is one-step satisfiable.

Lemma 2.10 (Named existence lemma, Version 1)**.** *If K is 0-pasted and $\mathcal{R}$ is strongly one-step complete, then there exists a named canonical K-model.*

In summary, we have:

Theorem 2.11. *If $\mathcal{R}$ is strongly one-step complete, then every extension of $\mathcal{LR}$ by pure axioms is both globally and locally strongly complete over countable hybrid models when equipped with the Name rule. That is, if $\Phi, \Psi \subseteq \mathcal{F}(\Lambda)$ and $\phi \in \mathcal{F}(\Lambda)$, then $\Phi; \Psi \vdash_{\mathcal{LRA}+\mathsf{Name}} \phi$ whenever $\Phi; \Psi \models^{\mathcal{C}} \phi$, where $\mathcal{C}$ is the class of countable $\mathcal{A}$-models.*

Proof. As usual, we show that every $(\mathcal{LRA} + \mathsf{Name})$-$\Phi$-consistent set $\Psi \subseteq \mathcal{F}(\Lambda)$ is satisfiable in a countable $\mathcal{A}$-model M such that $M \models \Phi$ (where satisfiability is clearly invariant under passing from $\mathcal{F}(\Lambda)$ to $\mathcal{F}(\Lambda)^+$). The extended Lindenbaum lemma yields a 0-pasted maximally consistent subset ABox $K \subseteq \mathcal{F}(\Lambda)^+$ and a nominal i in $\mathcal{F}(\Lambda)^+$ such that $@_i\Psi \subseteq K$. By the named existence lemma, we find a named, hence countable, canonical K-model $M = (S_K, \gamma, V_K)$, and by the truth lemma (Lemma 2.5), M is an $\mathcal{A}$-model, $M \models \Phi$, and $M, K_i \models \Psi$. ■

Remark 2.12. In the literature (e.g. [3, Theorem 7.29]), the above completeness theorem is sometimes phrased as "completeness with respect to named models", i.e. models where every state is the denotation of some nominal; such models also played a central role in the early development of hybrid logic by the Sofia school (see e.g. [15]). In detail, this means that every state of the model is the denotation of a nominal *in a language extended with countably many new nominals*. This extension is necessary, as otherwise the consistent set $\{\neg n \mid n \in \mathsf{N}\}$ would be satisfiable in a model where every state is named by a nominal $n \in \mathsf{N}$ of the original language, which is clearly impossible. Completeness with respect to models where every state is named by a nominal in an extended language, on the other hand, is an immediate consequence of completeness with respect to countable models.

Example 2.13. The previous theorem establishes strong completeness results for pure extensions of all hybrid logics with neighbourhood semantics (Example 1.1.4) that are defined by rank-1 axioms [20], i.e. modal formulas where the nesting depth of modalities is uniformly equal to 1 (such as the monotonicity axiom $\Box(a \wedge b) \to \Box b$). For the monotonic cases, i.e. extensions of monotonic hybrid logic, these results are essentially known [24], while they seem to be new for the non-monotonic cases, i.e. extensions of classical hybrid logic not containing the monotonicity axiom, including, e.g., various deontic logics [12]. Moreover, the theorem newly proves strong completeness of the hybridization of coalition logic, as Theorem 3.2 of [17] essentially states that coalition logic satisfies strong one-step completeness.

2.2. Pure Completeness for Bounded Logics

The condition of strong one-step completeness used in the previous section is a comparatively rare phenomenon [20]; the strength of the condition becomes clear in the fact that, unlike in the classical case of Kripke semantics, the above did not require a notion of 1-pastedness [5]. We proceed to present an alternative approach for the case where one does have an analogue of the (Paste-1) rule — this is the case if the operators are *bounded*, i.e., their satisfaction hinges, in each case, on only finitely and boundedly many states of a model.

Definition 2.14. A modal operator $\heartsuit$ is k*-bounded* for $k \in \mathbb{N}$ with respect to a Λ-structure T if for every set X and every $A \subseteq X$,

$$[\![\heartsuit]\!]_X(A) = \textstyle\bigcup_{B \subseteq A, \# B \leq k} [\![\heartsuit]\!]_X(B).$$

(This implies in particular that $\heartsuit$ is monotonic.) We say that Λ is bounded w.r.t. T if every modal operator $\heartsuit$ in Λ is $k_\heartsuit$-bounded for some $k_\heartsuit$.

The boundedness of an operator can now be internalized in the logical deduction system. In particular, for k-bounded operators $\heartsuit$, one has the *paste rule*

$$(\mathsf{Paste}\heartsuit(k))\quad \frac{@_{j_1}\phi \wedge \cdots \wedge @_{j_k}\phi \wedge @_i\heartsuit(j_1 \vee \cdots \vee j_k) \to \psi}{@_i\heartsuit\phi \to \psi}$$

with the side condition that the j_r are pairwise distinct fresh nominals. We write $\Phi \vdash_{\mathcal{LRA}+\mathsf{Name}+\mathsf{Paste}} \phi$ if ϕ is derivable from assumptions in Φ in the system $\mathcal{LR} + \mathsf{Name}$ where additionally the rule $(\mathsf{Paste}\heartsuit(k))$ may be used in deductions for k-bounded operators $\heartsuit$. This induces the notion of $(\mathcal{LRA}+\mathsf{Name}+\mathsf{Paste})$-$\Phi$-consistency, which we briefly refer to as consistency as we fix Φ, $\mathcal{A}$, and $\mathcal{R}$ throughout. Again, the system is clearly sound, i.e. $\Phi; \Psi \models^{\mathcal{C}} \phi$ whenever $\Phi; \Psi \vdash_{\mathcal{LRA}+\mathsf{Name}+\mathsf{Paste}} \phi$, where $\mathcal{C}$ is the class of $\mathcal{A}$-models.

Examples 2.15. 1. *Hybrid K.* The modal operator $\Diamond$ is 1-bounded. The arising paste rule $(\mathsf{Paste}\Diamond(1))$ is precisely the rule $(\mathit{paste}\Diamond)$ of [4].

2. *Graded hybrid logic.* The modal operator $\Diamond_k$ is $(k+1)$-bounded. One thus has a paste rule

$$(\mathsf{Paste}\Diamond_k(k+1))\quad \frac{@_{j_1}\phi \wedge \cdots \wedge @_{j_{k+1}}\phi \wedge @_i\Diamond_k(j_1 \vee \cdots \vee j_{k+1}) \to \psi}{@_i\Diamond_k\phi \to \psi}$$

with side conditions as before.

3. *Positive Presburger hybrid logic.* A Presburger operator $\sum a_i \cdot \#(\_i) \geq k$ (Example 1.1) is k-bounded if the a_i are positive. E.g., this still allows expressing the statement, generally believed to be valid in the German national football league, that a team that has at least 37 points will not be relegated: $3 \cdot \#\mathsf{win} + 1 \cdot \#\mathsf{draw} \geq 37 \to \neg\mathsf{relegated}$.

The generalized 1-pastedness condition for bounded operators is as follows.

Definition 2.16. Let Λ be bounded. An ABox K is 1-*pasted* if whenever $\heartsuit$ is k-bounded and $@_i\heartsuit\phi \in K$, then $\{@_{j_1}\phi, \dots, @_{j_k}\phi, @_i\heartsuit(j_1 \vee \dots \vee j_k)\} \subseteq K$ for some nominals $j_1, \dots, j_k$.

Again, it is clear that if Λ is bounded, then K can induce a named model only of K is 1-pasted. It is easy to see that if $\mathcal{R}$ is one-step complete and Λ is bounded (in fact already if $\mathcal{R}$ derives monotony for every $\heartsuit \in \Lambda$), then every 1-pasted set is also 0-pasted (Definition 2.6).

Lemma 2.17 (Extended Lindenbaum lemma for 1-pasted sets)**.** *Let Λ be bounded. If $\Psi \subseteq \mathcal{F}(\Lambda)$ is consistent, then there exist a* 1*-pasted maximally consistent ABox $K \subseteq @\mathcal{F}(\Lambda)^+$ and a nominal i in $\mathcal{F}(\Lambda)^+$ such that $@_i\Psi \subseteq K$, where $\mathcal{F}(\Lambda)^+$ is as in Section 2.1.*

Bounded operators now allow us to use a weaker version of one-step completeness. Instead of requiring that all one-step consistent sets are one-step satisfiable, we may restrict to *finite extensions* of propositional variables.

Definition 2.18. We say that $\mathcal{R}$ is *strongly finitary one-step complete* if for every set X, every one-step consistent subset of $\mathsf{Prop}(\Lambda(\mathcal{P}_{\mathit{fin}}(X)))$ is one-step satisfiable.

Clearly, any strongly one-step complete rule set is also strongly finitary one-step complete, but the example of graded hybrid logic witnesses that the converse is not true. We note that the weaker criterion still fails for probabilistic logics due to inherent non-compactness [23]; probabilistic logics also fail to be bounded, as a given probability $p \in [0, 1]$ can be split into any number of summands. Together with boundedness, the above condition enables a second version of the named existence lemma.

Lemma 2.19 (Named existence lemma, Version 2)**.** *If Λ is bounded, $\mathcal{R}$ is strongly finitary one-step complete, and K is* 1*-pasted, then there exists a named canonical K-model.*

Summarizing the above, we have the following extended completeness result.

Theorem 2.20. *Let Λ be bounded, and let $\mathcal{R}$ be strongly finitary one-step complete. Then every extension of $\mathcal{LR}$ by pure axioms is globally and locally strongly complete over countable hybrid models when equipped with the* Name *and* Paste *rules. In other words, if $\Phi, \Psi \subseteq \mathcal{F}(\Lambda)$, $\phi \in \mathcal{F}(\Lambda)$, and $\mathcal{C}$ is the class of all countable $\mathcal{A}$-models, then $\Phi; \Psi \vdash_{\mathcal{LRA}+\mathsf{Name}+\mathsf{Paste}} \phi$ whenever $\Phi; \Psi \models^{\mathcal{C}} \phi$.*

The proof follows the same route via extended Lindenbaum lemma, existence lemma, and truth lemma as for Theorem 2.11.

Example 2.21. By Example 2.15 and the fact that the known complete axiomatizations of the associated modal logics are in fact strongly finitary one-step complete, the previous theorem proves completeness of pure extensions of hybrid K, graded hybrid logic, and positive Presburger hybrid logic. Except for the standard case of hybrid K, these results seem to be new. In particular, we obtain completeness of pure extensions of graded (or positive Presburger) hybrid logic defining the following frame classes in multigraph semantics:

- The class of *Kripke frames*, seen as the class of multigraphs where the transition multiplicity between two individual states is always at most 1, defined by the pure axiom $\neg\Diamond_1 i$.

- The class of *reflexive* multigraphs, defined by the pure axiom $i \to \Diamond_0 i$.
- The class of *transitive* multigraphs, defined by the pure axioms $\Diamond_0 \Diamond_n i \to \Diamond_n i$, $n \geq 0$.
- The class of *symmetric* multigraphs, i.e., those where the transition multiplicity from x to y always equals the one from y to x, which is defined by the pure axioms $i \wedge \Diamond_k j \to @_j \Diamond_k i$.

Other frame classes of interest, see e.g. [3, Section 7.3], can be characterized similarly by translating the corresponding frame conditions from Kripke to multigraph semantics.

2.3. The Mixed Case

In some cases, the two methods laid out in the preceding sections can be combined for modal operators with several arguments that adhere, in each of their arguments, to one of the respective sets of semantic conditions. For the sake of readability, we formulate this explicitly only for the mixed binary case with a single modal operator, i.e. we assume in this section that $\Lambda = \{\heartsuit\}$ with $\heartsuit$ binary; the generalization to arbitrary numbers of arguments, several modal operators etc. should be obvious, and essentially only requires more elaborate terminology and notation.

Definition 2.22. We say that $\mathcal{R}$ is *(strongly, strongly finitary) one-step complete* if every one-step consistent subset of $\mathsf{Prop}(\Lambda(\mathcal{P}(X) \times \mathcal{P}_{\mathit{fin}}(X)))$ is one-step satisfiable. Moreover, we say that $\heartsuit$ is k*-bounded in the second argument* for $k \in \mathbb{N}$ if for every set X and all $A, B \subseteq X$, $[\![\heartsuit]\!]_X(A, B) = \bigcup_{C \subseteq A, \# C \leq k} [\![\heartsuit]\!]_X(A, C)$.

In the same manner as for Theorems 2.11 and 2.20, we derive:

Theorem 2.23. *If $\mathcal{R}$ is (strongly, strongly finitary) one-step complete and $\heartsuit$ is k-bounded in the second argument, then every extension of $\mathcal{LR}$ by pure axioms is both locally and globally strongly complete over countable hybrid models when equipped with the appropriate* Name *and* Paste *rules.*

Example 2.24. Hybrid CK (Example 1.1) is easily seen to be (strongly, strongly finitary) one-step complete, and the operator $>$ defined from the conditional operator $\Rightarrow$ by $a > b :\leftrightarrow \neg(a \Rightarrow \neg b)$ is 1-bounded in the second argument. By the above, it follows that every pure extension of hybrid CK is strongly complete over countable hybrid selection function models. E.g. we may define the class of conditional frames where all expressible conditions induce transitive relations by pure axioms $(\phi > \phi > i) \to (\phi > i)$. Such frames satisfy also the dual axiom (using a propositional variable a) $(\phi \Rightarrow a) \to (\phi \Rightarrow (\phi \Rightarrow a))$, an axiom for duplicating conditional assumptions. Similar statements apply to a combination of graded and conditional logic (obtainable compositionally using the methods of [21]), which has operators of the form $a \Rightarrow_k b$ "if a, then one normally has more than k instances of b".

The semantics of conditional logics in general has complex ramifications, involving, e.g., preference orderings or systems of spheres (see, e.g., [10, 18]); application of our methods to conditional logics beyond CK is the subject of further investigation. We note that pure completeness of a hybrid extension of Lewis' logic of counterfactuals has been established recently [18].

3. Local Binding

We next investigate completeness of a stronger hybrid language that includes the $\downarrow$ binder, which binds a state variable to the current state. Concretely, we allow formulas of the form $\downarrow i.\, \phi$, wherein the nominal i is locally bound (for compactness of presentation, we give up the usual distinction between nominals and state variables). Given a modal similarity type Λ, we write $\mathcal{F}_{\downarrow}(\Lambda)$ for the ensuing extension of $\mathcal{F}(\Lambda)$. The reading of the formula $\downarrow i.\phi$ is "ϕ holds for the current state i". The satisfaction relation in the extended logic is defined by an additional clause for the $\downarrow$ binder,

$$(C, \gamma, V) \models \downarrow i.\, \phi \text{ iff } (C, \gamma, V[c/i]) \models \phi$$

where c is a state in a coalgebra C and $V[c/i]$ is obtained from V by modifying the value of i to c. The semantics of the $\downarrow$ binder immediately translates into the axiom scheme (see e.g. [4])

$$(\mathsf{DA}) \qquad @_i((\downarrow j.\, \phi) \leftrightarrow \phi[i/j]).$$

Given a set $\mathcal{R}$ of Λ-rules, a set $\Phi \subseteq \mathcal{F}_{\downarrow}(\Lambda)$ of formulas and a set $\mathcal{A} \subseteq @\mathcal{F}_{\downarrow}(\Lambda)$ of pure axioms, we write $\Phi \vdash_{\mathcal{LRA}+\mathsf{Name}+\mathsf{Paste}+\mathsf{DA}} \phi$ for the extension of the associated provability predicate $\vdash_{\mathcal{LRA}+\mathsf{Name}+\mathsf{Paste}}$ with (DA). Using (DA), one easily proves an extension of the truth lemma for named models (Lemma 2.5) to $\mathcal{F}_{\downarrow}(\Lambda)$, so that the completeness results for pure extensions proved before (Theorems 2.11, 2.20, and 2.23) transfer immediately to $\mathcal{L}_{\downarrow}$. We make this explicit for the bounded case:

Theorem 3.1. *If Λ is bounded and $\mathcal{R}$ is strongly finitary one-step complete, then every pure extension of $\mathcal{L}_{\downarrow}$ is strongly locally and globally complete over countable hybrid models. In other words, $\Phi; \Psi \models^{\mathcal{C}} \phi$ iff $\Phi; \Psi \vdash_{\mathcal{LRA}+\mathsf{Name}+\mathsf{Paste}+\mathsf{DA}} \phi$ for all $\phi \in \mathcal{F}_{\downarrow}(\Lambda)$ and all $\Phi, \Psi \subseteq \mathcal{F}(\Lambda)$, where $\mathcal{C}$ is the class of all countable $\mathcal{A}$-models.*

Remark 3.2. As noted in [24], the named model construction more generally yields completeness for any *locally definable* extension of the hybrid language, i.e. any extension whose semantics *at named states* is defined by a formula similar to (DA).

Example 3.3. Continuing Example 2.15, Theorem 3.1 reproves not only the known completeness of pure extensions of hybrid K with $\downarrow$, but also the completeness of pure extensions of graded (or positive Presburger) hybrid logic with $\downarrow$. This extends easily to the multi-agent case, or, in description logic terminology, to description logics with multiple roles. As, moreover, both a role hierarchy and transitivity of roles can be defined using pure axioms, we thus arrive at a complete axiomatization of an extension of the description logic $\mathcal{SHOQ}$ with satisfaction operators and $\downarrow$, which has been used in connection with conjunctive query answering [11], and allows, e.g., talking about the number of stepchildren of a stepmother, in continuation of the stepmother example from [13], .

4. Conclusions

We have laid out two criteria for the existence of named canonical models in coalgebraic hybrid logics — one that applies to cases where one has an analogue of the so-called Paste-1 rule of standard hybrid logic, and one which applies to cases where one does not need any such rule. While the latter means essentially that the logic is equipped with a neighbourhood semantics, the former requires that all modal operators of the logic are bounded, i.e. there is always only a bounded number of states relevant for their satisfaction at each point. Our main novel example of this type is graded hybrid logic (and an extension of it using certain Presburger modalities [9]). The named model construction entails completeness of pure extensions and completeness of extended hybrid languages with the local binder $\downarrow$ (of which the I–me construct of [13] is a single-variable restriction), which we thus obtain as new results for, e.g., hybrid coalition logic, hybrid classical modal logic, several hybrid deontic logics, hybrid conditional logic, graded hybrid logic, and an extension of the description logic $\mathcal{SHOQ}$. An open question that remains is the existence of so-called orthodox axiomatizations [4] in the presence of $\downarrow$, as well as to find an analogue of the characterization result of [24] stating that a variant of the Paste-1 rule characterizes the Kripke models among the topological models of $S4$. A further topic of investigation is to find decidable fragments of the language with ↓; we note slightly speculatively that the fragment used in [13] may, in our terminology, be seen as requiring that a suitably defined NNF of a formula contains only positive occurrences of bound nominals under bounded modal operators.

References

[1] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In *Computer Science Logic, CSL 99*, vol. 1683 of *LNCS*, pp. 307–321, 1999.

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. *The Description Logic Handbook*. Cambridge University Press, 2003.

[3] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[4] P. Blackburn and B. ten Cate. Pure extensions, proof rules, and hybrid axiomatics. *Stud. Log.*, 84:277–322, 2006.

[5] P. Blackburn and M. Tzakova. Hybrid languages and temporal logic. *Logic J. IGPL*, 7:27–54, 1999.

[6] B. Chellas. *Modal Logic*. Cambridge University Press, 1980.

[7] C. Cirstea, A. Kurz, D. Pattinson, L. Schröder, and Y. Venema. Modal logics are coalgebraic. *The Computer Journal*, 2009. In print.

[8] G. D'Agostino and A. Visser. Finality regained: A coalgebraic study of Scott-sets and multisets. *Arch. Math. Logic*, 41:267–298, 2002.

[9] S. Demri and D. Lugiez. Presburger modal logic is only PSPACE-complete. In *Automated Reasoning, IJCAR 06*, vol. 4130 of *LNAI*, pp. 541–556. Springer, 2006.

[10] N. Friedman and J. Y. Halpern. On the complexity of conditional logics. In *Knowledge Representation and Reasoning, KR 94*, pp. 202–213. Morgan Kaufmann, 1994.

[11] B. Glimm, I. Horrocks, and U. Sattler. Conjunctive query answering for description logics with transitive roles. In *Description Logics, DL 06*, vol. 189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[12] L. Goble. A proposal for dealing with deontic dilemmas. In *Deontic Logic in Computer Science, DEON 04*, vol. 3065 of *LNAI*, pp. 74–113. Springer, 2004.

[13] M. Marx. Narcissists, stepmothers and spies. In *Description Logics, DL 02*, CEUR Workshop Proceedings, 2002.

[14] R. Myers, D. Pattinson, and L. Schröder. Coalgebraic hybrid logic. In *Foundations of Software Science and Computation Structures, FOSSACS 09*, LNCS. Springer, 2009. To appear.

[15] S. Passy and T. Tinchev. PDL with data constants. *Inf. Process. Lett.*, 20:35–41, 1985.

[16] D. Pattinson. Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. *Theoret. Comput. Sci.*, 309:177–193, 2003.

[17] M. Pauly. A modal logic for coalitional power in games. *J. Logic Comput.*, 12:149–166, 2002.

[18] K. Sano. Hybrid counterfactual logics. *J. Log. Lang. Inf.*, 18:515539, 2009.

[19] L. Schröder. A finite model construction for coalgebraic modal logic. *J. Log. Algebr. Prog.*, 73:97–110, 2007.

[20] L. Schröder and D. Pattinson. Rank-1 modal logics are coalgebraic. *J. Logic Comput.* In print.

[21] L. Schröder and D. Pattinson. Modular algorithms for heterogeneous modal logics. In *Automata, Languages and Programming, ICALP 07*, vol. 4596 of *LNCS*, pp. 459–471. Springer, 2007.

[22] L. Schröder and D. Pattinson. PSPACE bounds for rank-1 modal logics. *ACM Trans. Comput. Log.*, 10(2:13):1–33, 2009.

[23] L. Schröder and D. Pattinson. Strong completeness of coalgebraic modal logics. In *Theoretical Aspects of Computer Science, STACS 09*, Leibniz International Proceedings in Informatics, pp. 673–684. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2009.

[24] B. ten Cate and T. Litak. Topological perspective on the hybrid proof rules. In *Hybrid Logic, HyLo 06*, vol. 174 of *ENTCS*, pp. 79–94, 2007.

A DICHOTOMY THEOREM FOR THE GENERAL MINIMUM COST HOMOMORPHISM PROBLEM

RUSTEM TAKHANOV [1]

[1] Department of Computer and Information Science, Linköping University
E-mail address: g-rusta@ida.liu.se
E-mail address: takhanov@mail.ru

Abstract. In the constraint satisfaction problem (CSP), the aim is to find an assignment of values to a set of variables subject to specified constraints. In the minimum cost homomorphism problem ($MinHom$), one is additionally given weights c_{va} for every variable v and value a, and the aim is to find an assignment f to the variables that minimizes $\sum_v c_{vf(v)}$. Let $MinHom\,(\Gamma)$ denote the $MinHom$ problem parameterized by the set of predicates allowed for constraints. $MinHom\,(\Gamma)$ is related to many well-studied combinatorial optimization problems, and concrete applications can be found in, for instance, defence logistics and machine learning. We show that $MinHom\,(\Gamma)$ can be studied by using algebraic methods similar to those used for CSPs. With the aid of algebraic techniques, we classify the computational complexity of $MinHom\,(\Gamma)$ for all choices of Γ. Our result settles a general dichotomy conjecture previously resolved only for certain classes of directed graphs, [Gutin, Hell, Rafiey, Yeo, European J. of Combinatorics, 2008].

1. Introduction

Constraint satisfaction problems (CSP) are a natural way of formalizing a large number of computational problems arising in combinatorial optimization, artificial intelligence, and database theory. This problem has the following two equivalent formulations: (1) to find an assignment of values to a given set of variables, subject to constraints on the values that can be assigned simultaneously to specified subsets of variables, and (2) to find a homomorphism between two finite relational structures A and B. Applications of CSPs arise in the propositional logic, database and graph theory, scheduling and many other areas. During the past 30 years, CSP and its subproblems has been intensively studied by computer scientists and mathematicians. Considerable attention has been given to the case where the constraints are restricted to a given finite set of relations Γ, called a constraint language [3, 6, 13, 17]. For example, when Γ is a constraint language over the boolean set $\{0, 1\}$ with four ternary predicates $x \vee y \vee z$, $\overline{x} \vee y \vee z$, $\overline{x} \vee \overline{y} \vee z$, $\overline{x} \vee \overline{y} \vee \overline{z}$ we obtain 3-SAT. This direction of research has been mainly concerned with the computational complexity of $CSP\,(\Gamma)$ as a function of Γ. It has been shown that the complexity of $CSP\,(\Gamma)$ is highly

1998 ACM Subject Classification: F.4.1, G.2.2, I.2.6.

Key words and phrases: minimum cost homomorphisms problem, relational clones, constraint satisfaction problem, perfect graphs, supervised learning.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2493

connected with relational clones of universal algebra [13]. For every constraint language Γ, it has been conjectured that $CSP\left(\Gamma\right)$ is either in P or NP-complete [6].

In the minimum cost homomorphism problem ($MinHom$), we are given variables subject to constraints and, additionally, costs on variable/value pairs. Now, the task is not just to find any satisfying assignment to the variables, but one that minimizes the total cost.

Definition 1.1. Suppose we are given a finite domain set A and a finite constraint language $\Gamma \subseteq \bigcup\limits_{k=1}^{\infty} 2^{A^k}$. Denote by $MinHom\left(\Gamma\right)$ the following minimization task:

Instance: A first-order formula $\Phi\left(x_1,\ldots,x_n\right) = \bigwedge\limits_{i=1}^{N} \rho_i\left(y_{i1},\ldots,y_{in_i}\right),\ \rho_i \in \Gamma, y_{ij} \in \{x_1,\ldots,x_n\}$, and weights $w_{ia} \in \mathbb{N}, 1 \leq i \leq n, a \in A$.
Solution: Assignment $f : \{x_1,\ldots,x_n\} \to A$, that satisfies the formula Φ. If there is no such assignment, then indicate it.
Measure: $\sum\limits_{i=1}^{n} w_{if(x_i)}$.

Remark 1.2. Note that when we require weights to be positive we do not lose generality, since $MinHom\left(\Gamma\right)$ with arbitrary weights can be polynomial-time reduced to $MinHom\left(\Gamma\right)$ with positive weights by the following trick: we can add s to all weights, where s is some integer. This trick only adds ns to the value of the optimized measure. Hence, we can make all weights negative, and $MinHom\left(\Gamma\right)$ modified this way is equivalent to maximization but with positive weights only. This remark explains why both names $MinHom$ and $MaxHom$ can be allowed, though we prefer $MinHom$ due to historical reasons.

$MinHom$ was introduced in [11] where it was motivated by a real-world problem in defence logistics. The question for which directed graphs H the problem $MinHom\left(\{H\}\right)$ is polynomial-time solvable was considered in [8, 9, 10, 11, 12]. In this paper, we approach the problem in its most general form by algebraic methods and give a complete algebraic characterization of tractable constraint languages. From this characterization, we obtain a dichotomy for $MinHom$, i.e., if $MinHom\left(\Gamma\right)$ is not polynomial-time solvable, then it is NP-hard. Of course, this dichotomy implies the dichotomy for directed graphs.

In Section 2, we present some preliminaries together with results connecting the complexity of $MinHom$ with conservative algebras. The main dichotomy theorem is stated in Section 3 and its proof is divided into several parts which can be found in Sections 4-8. The NP-hardness results are collected in Section 4 followed by the building blocks for the tractability result: existence of majority polymorphisms (Section 5) and connections with optimization in perfect graphs (Section 6). Section 7 introduces the concept of *arithmetical deadlocks* which lays the foundation for the final proof in Section 8. Finally, in Section 9 we explain the relation of our results to previous research and present directions for future research.

2. Algebraic structure of tractable constraint languages

Recall that an optimization problem A is called NP-hard if some NP-complete language can be recognized in polynomial time with the aid of an oracle for A. We assume that $P \neq NP$.

Definition 2.1. Suppose we are given a finite set A and a constraint language $\Gamma \subseteq \bigcup\limits_{k=1}^{\infty} 2^{A^k}$. The language Γ is said to be *tractable* if, for every finite subset $\Gamma' \subseteq \Gamma$, $MinHom\,(\Gamma')$ is polynomial-time solvable, and Γ is called *NP-hard* if there is a finite subset $\Gamma' \subseteq \Gamma$ such that $MinHom\,(\Gamma')$ is NP-hard.

First, we will state some standard definitions from universal algebra.

Definition 2.2. Let $\rho \subseteq A^m$ and $f : A^n \to A$. We say that the function (operation) f *preserves* the predicate ρ if, for every $\left(x_1^i, \ldots, x_m^i\right) \in \rho, 1 \le i \le n$, we have that $\left(f\left(x_1^1, \ldots, x_1^n\right), \ldots, f\left(x_m^1, \ldots, x_m^n\right)\right) \in \rho$.

For a constraint language Γ, let $Pol\,(\Gamma)$ denote the set of operations preserving all predicates in Γ. Throughout the paper, we let A denote a finite domain and Γ a constraint language over A. We assume the domain A to be finite.

Definition 2.3. A constraint language Γ is called *a relational clone* if it contains every predicate expressible by a first-order formula involving only

a) predicates from $\Gamma \cup \left\{=^A\right\}$;
b) conjunction; and
c) existential quantification.

First-order formulas involving only conjunction and existential quantification are often called *primitive positive (pp) formulas.* For a given constraint language Γ, the set of all predicates that can be described by pp-formulas over Γ is called the *closure* of Γ and is denoted by $\langle\Gamma\rangle$.

For a set of operations F on A, let $Inv\,(F)$ denote the set of predicates preserved under the operations of F. Obviously, $Inv\,(F)$ is a relational clone. The next result is well-known [2, 7].

Theorem 2.4. *For a constraint language* Γ *over a finite set* A*,* $\langle\Gamma\rangle = Inv\,(Pol\,(\Gamma))$.

Theorem 2.4 tells us that the Galois closure of a constraint language Γ is equal to the set of all predicates that can be obtained via pp-formulas from the predicates in Γ.

Theorem 2.5. *For any finite constraint language* Γ *and any finite* $\Gamma' \subseteq \langle\Gamma\rangle$*, there is a polynomial time reduction from* $MinHom\,(\Gamma')$ *to* $MinHom\,(\Gamma)$.

Proof. Since any predicate from Γ' can be viewed as a pp-formula with predicates in Γ, an input formula to $MinHom\,(\Gamma')$ can be represented on the form $\Phi\,(x_1, \ldots, x_n) = \bigwedge\limits_{i=1}^{N} \exists z_{i1}, \ldots, z_{im_i} \Phi_i\,(y_{i1}, \ldots, y_{in_i}, z_{i1}, \ldots, z_{im_i})$, where $y_{ij} \in \{x_1, \ldots, x_n\}$ and Φ_i is a first-order formula involving only predicates in Γ, equality, and conjunction. Obviously, this formula is equivalent to $\exists z_{11}, \ldots, z_{Nm_N} \bigwedge\limits_{i=1}^{N} \Phi_i\,(y_{i1}, \ldots, y_{in_i}, z_{i1}, \ldots, z_{im_i})$. $\bigwedge\limits_{i=1}^{N} \Phi_i\,(y_{i1}, \ldots, y_{in_i}, z_{i1}, \ldots, z_{im_i})$ can be considered as an instance of $MinHom\left(\Gamma \cup \left\{=^A\right\}\right)$ with variables $x_1, \ldots, x_n, z_{11}, \ldots, z_{Nm_N}$ where weights w_{ij} will remain the same and for additional variables z_{kl} we define $w_{z_{kl}j} = 0$. By solving $MinHom\left(\Gamma \cup \left\{=^A\right\}\right)$ with the described input, we can find a solution of the initial $MinHom\,(\Gamma')$ problem. It is easy to see that the number of added variables is bounded by a polynomial in n. So this reduction can be carried out in polynomial time. Finally, $MinHom\left(\Gamma \cup \left\{=^A\right\}\right)$ can be reduced polynomially to $MinHom\,(\Gamma)$ because an equality constraint for a pair of variables is equivalent to replacement of all inclusions of the first variable in a formula by the second one. ■

The previous theorem tells us that the complexity of $MinHom\,(\Gamma)$ is basically determined by $Inv\,(Pol\,(\Gamma))$, i.e., by $Pol\,(\Gamma)$. That is why we will be concerned with the classification of sets of operations F for which $Inv\,(F)$ is a tractable constraint language.

Definition 2.6. An *algebra* is an ordered pair $\mathbb{A} = (A, F)$ such that A is a nonempty set (called a universe) and F is a family of finitary operations on A. An algebra with a finite universe is referred to as a finite algebra.

Definition 2.7. An algebra $\mathbb{A} = (A, F)$ is called *tractable* if $Inv(F)$ is a tractable constraint language and $\mathbb{A}$ is called *NP-hard* if $Inv(F)$ is an NP-hard constraint language.

In the following theorem, we show that we only need to consider a very special type of algebras, so called *conservative* algebras.

Definition 2.8. An algebra $\mathbb{A} = (A, F)$ is called conservative if for every operation $f \in F$ we have that $f\,(x_1, \ldots, x_n) \in \{x_1, \ldots, x_n\}$.

Theorem 2.9. *For any finite constraint language Γ over A and $C \subseteq A$, there is a polynomial time Turing reduction from $MinHom\,(\Gamma \cup \{C\})$ to $MinHom\,(\Gamma)$.*

Proof. Let the first-order formula $\Phi\,(x_1, \ldots, x_n) = \overset{M}{\underset{i=1}{\wedge}} C\,(y_i) \wedge \overset{N}{\underset{i=1}{\wedge}} \rho_i\,(z_{i1}, \ldots, z_{in_i})$, where $\rho_i \in \Gamma, y_i, z_{ij} \in \{x_1, \ldots, x_n\}$, and weights $w_{ia}, 1 \leq i \leq n, a \in A$ be an instance of $MinHom\,(\Gamma \cup \{C\})$. We assume without loss of generality that $y_i \neq y_j$, when $i \neq j$. Let $W = \sum\limits_{i=1}^{n} \sum\limits_{a \in A} w_{ia} + 1$ and define a new formula and weights

$$\Phi'\,(x_1, \ldots, x_n) = \overset{N}{\underset{i=1}{\wedge}} \rho_i\,(z_{i1}, \ldots, z_{in_i})$$

$$w'_{ia} = \begin{cases} w_{ia} + W, \text{if } a \notin C, \exists j\ x_i = y_j \\ w_{ia}, \text{ otherwise} \end{cases}$$

Then, using an oracle for $MinHom\,(\Gamma)$, we can solve

$$\min_{f \text{ satisfies } \Phi'} \sum_j w'_{jf(x_j)}.$$

Suppose that $\Phi\,(x_1, \ldots, x_n)$ is satisfiable and f is a satisfying assignment. It is easy to see that the part of the measure $\sum\limits_j w'_{jf(x_j)}$ that corresponds to the added values W is equal to 0 and the measure cannot be greater than $W - 1$. If g is any assignment that does not satisfy $\overset{M}{\underset{i=1}{\wedge}} C\,(y_i)$, then we see that this part of measure cannot be 0, and hence, is greater or equal to W. This means that the minimum in the task is achieved on satisfying assignments of $\Phi\,(x_1, \ldots, x_n)$ and any such assignment minimize the part of the measure that corresponds to the initial weights, i.e., $\sum\limits_i w_{if(x_i)}$.

If $\Phi\,(x_1, \ldots, x_n)$ is not satisfiable, then either Φ' is not satisfiable or $\min\limits_{f \text{ satisfies } \Phi'} \sum\limits_j w'_{jf(x_j)} \geq W$. Using an oracle for $MinHom\,(\Gamma)$, we can easily check this.

Consequently, $MinHom\,(\Gamma \cup \{C\})$ is polynomial-time reducible to $MinHom\,(\Gamma)$. ∎

Theorem 2.10. *If Γ is a constraint language over A that contains all unary relations, then $\mathbb{A} = (A, Pol\,(\Gamma))$ is conservative.*

Proof. Let $C = \{x_1, \ldots, x_n\} \subseteq A$. If a function $f : A^n \to A$ preserves the predicate C, then $f\,(x_1, \ldots, x_n) \in \{x_1, \ldots, x_n\}$. ■

3. Structure of tractable conservative algebras

Let $g : A^k \to A$ be an arbitrary conservative function and $S \subseteq A$. Define the function $g|_S : S^k \to S$, such that $\forall x_1, \ldots, x_k \in S\ g|_S\,(x_1, \ldots, x_k) = g\,(x_1, \ldots, x_k)$, i.e. the restriction of g to the set S. Throughout this paper we will consider a conservative algebra (A, F). For every $B \subseteq A$, let $F|_B = \{f_B | f \in F\}$. We assume that F is closed under superposition and variable change and contains all projections, i.e., it is *a functional clone*, because closing the set F under these operations does not change the set $Inv\,(F)$.

Sometimes we will consider clones as algebras and to describe them we will use the terms (conservativeness, tractability, NP-hardness) defined for algebras. All tractable clones, in case $A = \{0, 1\}$, can be easily found using well-known classification of boolean clones [15].

Theorem 3.1. *The boolean functional clone H is tractable if either $\{x \wedge y, x \vee y\} \subseteq H$ or $\{(x \wedge \overline{y}) \vee (\overline{y} \wedge z) \vee (x \wedge z)\} \subseteq H$, where $\wedge, \vee$ denote conjunction and disjunction. Otherwise, H is NP-hard.*

Every 2-element subalgebra of a tractable algebra must be tractable, which motivates the following definition.

Definition 3.2. Let F be a conservative functional clone. We say that F satisfies the *necessary local conditions* if and only if for every 2-element subset $B \subseteq A$, either

(1) there exists $f^{\wedge}, f^{\vee} \in F$ s.t. $f^{\wedge}|_B$ and $f^{\vee}|_B$ are different binary commutative functions; or
(2) there exists $f \in F$ s.t. $f|_B\,(x, x, y) = f|_B\,(y, x, x) = f|_B\,(y, x, y) = y$.

Theorem 3.3. *Suppose F is a conservative functional clone. If F is tractable, then it satisfies the necessary local conditions. If F does not satisfy the necessary local conditions, then it is NP-hard.*

In general, the necessary local conditions are not sufficient for tractability of a conservative clone. Let $M = \{B | B \subseteq A, |B| = 2, F|_B \text{ contains different binary commutative functions}\}$ and $\overline{M} = \{B | B \subseteq A, |B| = 2\} \setminus M$.

Suppose $f \in F$. By $\overset{a}{\underset{b}{\downarrow}} f$ we mean $a \neq b$ and $f\,(a, b) = f\,(b, a) = b$. For example, $\overset{1}{\underset{2}{\downarrow}}\overset{2}{\underset{3}{\downarrow}}\overset{1}{\underset{3}{\downarrow}} f$ means that $f|_{\{1,2,3\}}\,(x, y) = \max\,(x, y)$.

Introduce an undirected graph without loops $T_F = (M^o, P)$ where $M^o = \{(a, b) \,|\, \{a, b\} \in M\}$ and $P = \left\{ \langle (a, b), (c, d) \rangle \,|\, (a, b), (c, d) \in M^o, \text{ there is no } f \in F : \overset{a}{\underset{b}{\downarrow}}\overset{c}{\underset{d}{\downarrow}} f \right\}$. The core result of the paper is the following.

Theorem 3.4. *Suppose F satisfy the necessary local conditions. If the graph $T_F = (M^o, P)$ is bipartite, then F is tractable. Otherwise, F is NP-hard.*

The proof of this theorem will be given in two steps. Firstly, in the following section, we will prove NP-hardness of F when $T_F = (M^o, P)$ is not bipartite. The final sections will be dedicated to the polynomial-time solvable cases.

4. NP-hard case

In this section, we will prove that if a set of functions F satisfies the necessary local conditions and $T_F = (M^o, P)$ (as defined in the previous section) is not bipartite, then F is NP-hard. Let ${}^{a}_{b}\overline{\times}{}^{c}_{d}$ and ${}^{a}_{b}\times{}^{c}_{d}$ denote the predicates $\{a, b\} \times \{c, d\} \setminus \{(b, d)\}$ and $\{(a, d), (b, c)\}$, where $a \neq b, c \neq d$. We need the following lemmas.

Lemma 4.1. *A constraint language that contains* $\left\{ {}^{a_0}_{b_0}\overline{\times}{}^{a_1}_{b_1}, \dots, {}^{a_{2k-1}}_{b_{2k-1}}\overline{\times}{}^{a_{2k}}_{b_{2k}}, {}^{a_{2k}}_{b_{2k}}\overline{\times}{}^{a_0}_{b_0} \right\}$ *is NP-hard.*

Lemma 4.2. *If* $\langle (a, b), (c, d) \rangle \in P$*, then either* ${}^{a}_{b}\overline{\times}{}^{c}_{d} \in Inv\,(F)$*, or* ${}^{a}_{b}\times{}^{c}_{d} \in Inv\,(F)$*.*

Proof of NP-hard case of Theorem 3.4. For binary predicates α, β, let $\alpha \circ \beta = \{(x, y) | \exists z : \alpha(x, z) \wedge \beta(z, y)\}$. Obviously, if $\alpha, \beta \in Inv\,(F)$, then $\alpha \circ \beta \in Inv\,(F)$, too.

Since $T_F = (M^o, P)$ is not bipartite, we can find a shortest odd cycle in it, i.e. a sequence $(a_0, b_0), (a_1, b_1), \dots, (a_{2k}, b_{2k}) \in M^o, k \geq 1$, such that $\langle (a_i, b_i), (a_{i\oplus 1}, b_{i\oplus 1}) \rangle \in P$. Here, $i \oplus j$ denotes $i + j (\mathrm{mod}\ 2k + 1)$.

By Lemma 4.2, there is a cyclic sequence $\rho_{0,1}, \rho_{1,2}, \dots, \rho_{2k,0} \in Inv\,(F)$ such that $\rho_{i,i\oplus 1}$ is either equal to ${}^{a_i}_{b_i}\overline{\times}{}^{a_{i\oplus 1}}_{b_{i\oplus 1}}$ or equal to ${}^{a_i}_{b_i}\times{}^{a_{i\oplus 1}}_{b_{i\oplus 1}}$. Note that all predicates cannot be of the second type: otherwise, we have $\rho_{0,1} \circ \rho_{1,2} \circ \dots \circ \rho_{2k,0} = {}^{a_0}_{b_0}\times{}^{a_0}_{b_0}$ which contradicts that $\{a_0, b_0\} \in M$.

If the sequence contains a fragment $\rho_{i,i\oplus 1} = {}^{a_i}_{b_i}\times{}^{a_{i\oplus 1}}_{b_{i\oplus 1}}$, $\rho_{i\oplus 1,i\oplus 2} = {}^{a_{i\oplus 1}}_{b_{i\oplus 1}}\times{}^{a_{i\oplus 2}}_{b_{i\oplus 2}}$, $\rho_{i\oplus 2,i\oplus 3} = {}^{a_{i\oplus 2}}_{b_{i\oplus 2}}\overline{\times}{}^{a_{i\oplus 3}}_{b_{i\oplus 3}}$, then these predicates can be replaced by:

$$\rho_{i,i\oplus 3} \triangleq \rho_{i,i\oplus 1} \circ \rho_{i\oplus 1,i\oplus 2} \circ \rho_{i\oplus 2,i\oplus 3} = {}^{a_i}_{b_i}\times{}^{a_{i\oplus 1}}_{b_{i\oplus 1}} \circ {}^{a_{i\oplus 1}}_{b_{i\oplus 1}}\times{}^{a_{i\oplus 2}}_{b_{i\oplus 2}} \circ {}^{a_{i\oplus 2}}_{b_{i\oplus 2}}\overline{\times}{}^{a_{i\oplus 3}}_{b_{i\oplus 3}} = {}^{a_i}_{b_i}\overline{\times}{}^{a_{i\oplus 3}}_{b_{i\oplus 3}}$$

Let us replace $\rho_{i,i\oplus 1}$, $\rho_{i\oplus 1,i\oplus 2}$, $\rho_{i\oplus 2,i\oplus 3}$ by $\rho_{i,i\oplus 3}$ in the sequence $\rho_{0,1}, \rho_{1,2}, \dots, \rho_{2k,0}$. We have $\langle (a_i, b_i), (a_{i\oplus 3}, b_{i\oplus 3}) \rangle \in P$, since otherwise the predicate $\rho_{i,i\oplus 3}$ is not preserved. Hence, we can delete two vertices in the cycle $(a_0, b_0), (a_1, b_1), \dots, (a_{2k}, b_{2k}) \in M^o$. This contradicts that this sequence is the shortest among odd sequences. Therefore, such a fragment does not exist.

If the sequence contains a fragment $\rho_{i,i\oplus 1} = {}^{a_i}_{b_i}\overline{\times}{}^{a_{i\oplus 1}}_{b_{i\oplus 1}}$, $\rho_{i\oplus 1,i\oplus 2} = {}^{a_{i\oplus 1}}_{b_{i\oplus 1}}\times{}^{a_{i\oplus 2}}_{b_{i\oplus 2}}$, $\rho_{i\oplus 2,i\oplus 3} = {}^{a_{i\oplus 2}}_{b_{i\oplus 2}}\overline{\times}{}^{a_{i\oplus 3}}_{b_{i\oplus 3}}$, then these predicates can be replaced by:

$$\rho_{i,i\oplus 3} \triangleq \rho_{i,i\oplus 1} \circ \rho_{i\oplus 1,i\oplus 2} \circ \rho_{i\oplus 2,i\oplus 3} = {}^{a_i}_{b_i}\overline{\times}{}^{a_{i\oplus 1}}_{b_{i\oplus 1}} \circ {}^{a_{i\oplus 1}}_{b_{i\oplus 1}}\times{}^{a_{i\oplus 2}}_{b_{i\oplus 2}} \circ {}^{a_{i\oplus 2}}_{b_{i\oplus 2}}\overline{\times}{}^{a_{i\oplus 3}}_{b_{i\oplus 3}} = {}^{a_i}_{b_i}\overline{\times}{}^{a_{i\oplus 3}}_{b_{i\oplus 3}}$$

As in the previous case, we obtain a contradiction. Consequently, we have an odd sequence ${}^{a_0}_{b_0}\overline{\times}{}^{a_1}_{b_1}, {}^{a_1}_{b_1}\overline{\times}{}^{a_2}_{b_2}, \dots, {}^{a_{2k-1}}_{b_{2k-1}}\overline{\times}{}^{a_{2k}}_{b_{2k}}, {}^{a_{2k}}_{b_{2k}}\overline{\times}{}^{a_0}_{b_0} \in Inv\,(F)$. By Lemma 4.1, this class of predicates is NP-hard. ■

5. Existence of the majority operation

The necessary local conditions tell that every two-element subalgebra of a tractable algebra contains certain operations. The simplest algebras over a domain A that satisfy these conditions are the following: $F_1 = \{\phi, \psi\}$ where ϕ, ψ are conservative commutative operations such that $\phi(a,b) \neq \psi(a,b)$ for every $a \neq b \in A$, and $F_2 = \{m\}$ where m is a conservative arithmetical operation, i.e. $m\left(x,x,y\right) = m\left(y,x,x\right) = m\left(y,x,y\right) = y$. This leads us to the following definitions.

Definition 5.1. Suppose a set of operations H over D is conservative and $B \subseteq \{\{x,y\} \,| x,y \in D, x \neq y\}$. A pair of binary operations $\phi, \psi \in H$ is called a *tournament pair* on B, if $\forall \{x,y\} \in B$ $\phi\left(x,y\right) = \phi\left(y,x\right), \psi\left(x,y\right) = \psi\left(y,x\right), \phi\left(x,y\right) \neq \psi\left(x,y\right)$ and for arbitrary $\{x,y\} \in \overline{B}$, $\phi\left(x,y\right) = x, \psi\left(x,y\right) = x$. An operation $m \in H$ is called *arithmetical* on B, if $\forall \{x,y\} \in B$ $m\left(x,x,y\right) = m\left(y,x,x\right) = m\left(y,x,y\right) = y$.

Definition 5.2. An operation $\mu : A^3 \to A$, satisfying the equality

$$\mu\left(x,y,y\right) = \mu\left(y,x,y\right) = \mu\left(y,y,x\right) = y$$

is called majority operation.

Theorem 5.3. *If F satisfies the necessary local conditions and $T_F = (M^o, P)$ is bipartite, then F contains a tournament pair on M.*

Proof. Let M_1, M_2 denote a partitioning of the bipartite graph $T_F = (M^o, P)$. Then, for every $\left(a,b\right), \left(c,d\right) \in M_1$, there is a function $\phi \in F : \overset{a}{\underset{b}{\downarrow}}\overset{c}{\underset{d}{\downarrow}}\phi$. Let us prove by induction that for every $\left(a_1,b_1\right), \left(a_2,b_2\right), \ldots, \left(a_n,b_n\right) \in M_1$, there is a $\phi : \overset{a_1}{\underset{b_1}{\downarrow}}\overset{a_2}{\underset{b_2}{\downarrow}}\ldots\overset{a_n}{\underset{b_n}{\downarrow}}\phi$.

The base of induction $n = 2$ is obvious. Let $\left(a_1,b_1\right), \left(a_2,b_2\right), \ldots, \left(a_{n+1},b_{n+1}\right) \in M_1$ be given. By the induction hypothesis, there are $\phi_1, \phi_2, \phi_3 \in F$: $\overset{a_2}{\underset{b_2}{\downarrow}}\ldots\overset{a_n}{\underset{b_n}{\downarrow}}\overset{a_{n+1}}{\underset{b_{n+1}}{\downarrow}}\phi_1, \overset{a_1}{\underset{b_1}{\downarrow}}\overset{a_3}{\underset{b_3}{\downarrow}}\ldots\overset{a_n}{\underset{b_n}{\downarrow}}\overset{a_{n+1}}{\underset{b_{n+1}}{\downarrow}}\phi_2, \overset{a_1}{\underset{b_1}{\downarrow}}\overset{a_2}{\underset{b_2}{\downarrow}}\ldots\overset{a_n}{\underset{b_n}{\downarrow}}\phi_3$. Then, it is easy to see that $\overset{a_1}{\underset{b_1}{\downarrow}}\ldots\overset{a_n}{\underset{b_n}{\downarrow}}\overset{a_{n+1}}{\underset{b_{n+1}}{\downarrow}}\phi_3\left(\phi_1\left(x,y\right), \phi_2\left(x,y\right)\right)$ which completes the induction proof.

The analogous statement can be proved for M_2. Moreover, $M_2 = \{(x,y) \,|\, (y,x) \in M_1\}$. So it follows from the proof that there are binary operations $\phi', \psi' \in F$, such that $\forall \left(x,y\right) \in M_1$: $\overset{x}{\underset{y}{\downarrow}}\phi'$ and $\forall \left(x,y\right) \in M_2$: $\overset{x}{\underset{y}{\downarrow}}\psi'$. Thus, the operations $\phi\left(x,y\right) = \phi'\left(x, \phi'\left(y,x\right)\right)$ and $\psi\left(x,y\right) = \psi'\left(x, \psi'\left(y,x\right)\right)$ satisfy the conditions of theorem. ∎

The proof of the following theorem uses the ideas from [3].

Theorem 5.4. *If F satisfies the necessary local conditions and $\overline{M} \neq \emptyset$, then F contains an arithmetical operation on $\overline{M}$.*

Theorem 5.5. *If F satisfies the necessary local conditions and $T_F = (M^o, P)$ is bipartite, then F contains a majority operation μ.*

Proof. If $\overline{M} \neq \emptyset$, then by Theorem 5.4, F contains a function $m : A^3 \to A$ that is arithmetical on $\overline{M}$. Then the function $\mu^1\left(x,y,z\right) = m\left(x, m\left(x,y,z\right), z\right)$ satisfies the conditions $\forall \{x,y\} \in \overline{M}$ $\mu^1\left(x,y,y\right) = \mu^1\left(y,x,y\right) = \mu^1\left(y,y,x\right) = y$. It is clear that, in the case where $M = \emptyset$, we can take μ^1 as majority μ.

If $M \neq \emptyset$, then by Theorem 5.3, there is a tournament pair $\phi, \psi : A^2 \to A$ on M. Then, the function $\mu^2(x,y,z) = \phi(\phi(\psi(x,y), \psi(y,z)), \psi(x,z))$ satisfies conditions $\forall \{x,y\} \in M\ \mu^2(x,y,y) = \mu^2(y,x,y) = \mu^2(y,y,x) = y$, and $\forall \{x,y,z\} \in \overline{M}\ \mu^2(x,y,z) = x$. If $\overline{M} = \emptyset$, then we can take μ^2 as the majority μ.

Finally, if $M, \overline{M} \neq \emptyset$, then $\mu(x,y,z) = \mu^1\left(\mu^2(x,y,z), \mu^2(y,z,x), \mu^2(z,x,y)\right)$. ■

6. Consistency and microstructure graphs

Every predicate in $Inv(F)$, when F contains a majority operation, is equal to the join of its binary projections [1]. To prove Theorem 3.4, it is consequently sufficient to prove polynomial-time solvability of $MinHom(\Gamma)$ where $\Gamma = \left\{\rho | \rho \subseteq A^2, \rho \in Inv(F)\right\}$, i.e. the $MinHom$ problem restricted to binary constraint languages.

Definition 6.1. Suppose we are given a constraint language Γ over A. Denote by $2 - MinHom(\Gamma)$ the following minimization problem:
Instance: A finite set of variables $X = \{x_1, \ldots, x_n\}$, a constraints pair (U, B) where $U = \langle \rho_i \rangle_{1 \leq i \leq n}$, $B = \langle \rho_{kl} \rangle_{1 \leq k \neq l \leq n}$, $\rho_i, \rho_{kl} \in \Gamma$, and weights $w_{ia}, 1 \leq i \leq n, a \in A$.
Solution: Assignment $f : \{x_1, \ldots, x_n\} \to A$, such that $\forall i\ f(x_i) \in \rho_i$ and $\forall k \neq l\ \ (f(x_k), f(x_l)) \in \rho_{kl}$.
Measure: $\sum\limits_{i=1}^{n} w_{if(x_i)}$.

We suppose everywhere that $\rho_{kl} = \rho_{lk}^t$ (where $\rho^t = \{(b,a) \,|\, (a,b) \in \rho\}$). If $\rho_{kl} \neq \rho_{lk}^t$, then we can always define $\forall k \neq l\ \rho_{kl} := \rho_{kl} \cap \rho_{lk}^t$, which does not change the set $\{(a,b) \,|\, (a,b) \in \rho_{kl}, (b,a) \in \rho_{lk}\}$. For a binary predicate ρ, define projections $\mathrm{Pr}_1\, \rho = \{a | (a,b) \in \rho\}$ and $\mathrm{Pr}_2\, \rho = \{b | (a,b) \in \rho\}$.

Definition 6.2. An instance of $2 - MinHom(\Gamma)$ with constraints pair $U = \langle \rho_i \rangle_{1 \leq i \leq n}$, $B = \langle \rho_{kl} \rangle_{1 \leq k \neq l \leq n}$ is called *arc-consistent* if $\forall i \neq j : \mathrm{Pr}_1\, \rho_{ij} = \rho_i, \mathrm{Pr}_2\, \rho_{ij} = \rho_j$ and is called *path-consistent* if for each different $i, j, k : \rho_{ik} \subseteq \rho_{ij} \circ \rho_{jk}$.

Obviously, by applying operations of the type $\rho_i := \rho_i \cap \mathrm{Pr}_1\, \rho_{ij}$, $\rho_j := \rho_j \cap \mathrm{Pr}_2\, \rho_{ij}$, $\rho_{ij} := \rho_{ij} \cap (\rho_i \times A)$, $\rho_{ij} := \rho_{ij} \cap (A \times \rho_j)$, $\rho_{ik} := \rho_{ik} \cap (\rho_{ij} \circ \rho_{jk})$, we can always make an instance arc-consistent and path-consistent in polynomial time. It is clear that under this transformations the set of feasible solutions does not change.

Definition 6.3. The *microstructure graph* [14] of an instance of $2 - MinHom(\Gamma)$ with constraints pair $U = \langle \rho_i \rangle_{1 \leq i \leq n}$, $B = \langle \rho_{kl} \rangle_{1 \leq k \neq l \leq n}$ is the graph $M_{U,B} = (V, E)$, where $V = \{(i,a) \,| 1 \leq i \leq n, a \in \rho_i\}$ and $E = \{\langle (i,a), (j,b) \rangle \,| i \neq j, (a,b) \in \rho_{ij}\}$.

Theorem 6.4. *Let $I = (X, U, B, w)$ be a satisfiable instance of $2 - MinHom(\Gamma)$. Then there is a one-to-one correspondence between maximal-size cliques of $M_{U,B}$ and satisfying assignments of I.*

Proof. The microstructure graph of an instance with constraints pair $U = \langle \rho_i \rangle_{1 \leq i \leq n}$, $B = \langle \rho_{kl} \rangle_{1 \leq k \neq l \leq n}$ is, obviously, n-partite, since $V = \bigcup\limits_{i=1}^{n} \{i\} \times \rho_i$ and pairs $(i,a), (i,b), a \neq b$ are not connected. Therefore, the cardinality of a maximal clique of $M_{U,B} = (V, E)$ is not greater than n.

If the cardinality of a maximal clique $S \subseteq V$ is n, then, for every i, $|S \cap (\{i\} \times \rho_i)| = 1$. Then, denoting the only element of $S \cap (\{i\} \times \rho_i)$ by v_i, we see that the assignment $f(x_i) = v_i$ satisfies all constraints. The opposite is also true, i.e., if the constraints $\langle \rho_i \rangle_{1 \le i \le n}, \langle \rho_{kl} \rangle_{1 \le k \ne l \le n}$ can be satisfied by some assignment f, then $\{(i, f(x_i)) \,|1 \le i \le n\}$ is a clique of cardinality n. ■

Hence, $2 - MinHom(\Gamma)$ can be reduced to finding a maximal-size clique $S \subseteq V$ of a microstructure graph that minimizes the following value:

$$\sum_{(i,a) \in S} w_{ia}.$$

Definition 6.5. Let $MMClique$ (Minimal weight among maximal-size cliques) denote the following minimization problem:
Instance: A graph $G = (V, E)$ and weights $w_i \in \mathbb{N}, i \in V$.
Solution: A maximal-size clique $K \subseteq V$ of G.
Measure: $\sum\limits_{v \in K} w_v$.

The following theorem connects perfect microstructure graphs and the complexity of $MinHom$.

Theorem 6.6. *Suppose we are given a class of conservative functions F containing a majority operation. If the microstructure graph is perfect for arbitrary arc-consistent and path-consistent instances of* $2 - MinHom(Inv(F))$*, then F is tractable.*

Definition 6.7. A cycle C_{2k+1}, $k \ge 2$, is called *an odd hole* and its complement graph *an odd antihole.*

In Section 8 we will use the following conjecture of Berge, which was proved in [4].

Theorem 6.8. *A graph is perfect if and only if it does not contain an induced subgraph isomorphic to an odd hole or antihole.*

We say that a graph is *of the type* $S_{2k+1}, k \ge 2$ if it is isomorphic to the graph with vertex set $\{0, 1, \dots, 2k\}$, where vertices $i \,(\text{mod } 2k+1)$, $i+1 \,(\text{mod } 2k+1)$ are not connected and vertices $i \,(\text{mod } 2k+1)$, $i+2 \,(\text{mod } 2k+1)$ are connected. Other pairs can be connected arbitrarily. Obviously, every odd hole or antihole is of one of the types $S_{2k+1}, k \ge 2$.

7. Arithmetical deadlocks

The key idea for the proof of the polynomial case of Theorem 3.4 is to show that path- and arc-consistent instances of $2 - MinHom(Inv(F))$ have a perfect microstructure graph. We will prove this by showing that the microstructure graph forbids certain types of subgraphs. The exact formulation of the result can be found below in Theorem 8.1. This theorem uses the nonexistence of structures called *arithmetical deadlocks* which are introduced in this section.

Definition 7.1. Suppose H is a conservative set of functions over D, $m \in H$ is an arithmetical operation on $B \subseteq \{\{x, y\} \,|x, y \in D, x \ne y\}$ and the pair $\phi, \psi \in H$ is a tournament pair on $\overline{B}$. An instance of $2 - MinHom(Inv(H))$ with constraints pair $U = \langle \rho_i \rangle_{1 \le i \le n}$, $B = \langle \rho_{kl} \rangle_{1 \le k \ne l \le n}$ is called *an odd arithmetical deadlock* if there is a subset $\{i_0, \dots, i_{k-1}\} \subseteq \{1, \dots, n\}, k \ge 3$ of odd cardinality and $\{x_0, y_0\}, \dots, \{x_{k-1}, y_{k-1}\} \in B$, such that for $0 \leq s \leq k-1$: $\rho_{i_s, i_{s\oplus 1}} \cap \{x_s, y_s\} \times \{x_{s\oplus 1}, y_{s\oplus 1}\} = {}^{x_s}_{y_s}\!\times\!{}^{x_{s\oplus 1}}_{y_{s\oplus 1}}$, where $i \oplus j$ denotes $i + j (\mathrm{mod}\ k)$. The subset $\{i_0, \dots, i_{k-1}\}$ is called *a deadlock subset.*

Theorem 7.2. *Suppose H is a conservative set of functions over D, $m \in H$ is an arithmetical operation on $B \subseteq \{\{x,y\} \,|x, y \in D, x \neq y\}$ and the pair $\phi, \psi \in H$ is a tournament pair on $\overline{B}$. If an instance of $2 - MinHom\,(Inv\,(H))$ is arc-consistent and path-consistent, then it cannot be an odd arithmetical deadlock.*

8. Final step in a proof of polynomial case

Theorem 8.1. *Suppose that F satisfies the necessary local conditions and that the graph $T_F = (M^o, P)$ is bipartite. Then for every path- and arc-consistent instance of $2 - MinHom\,(Inv\,(F))$, its microstructure graph forbids subgraphs of the type $S_{2p+1}, p \geq 2$.*

Proof. Suppose to the contrary that we have a path- and arc-consistent instance $I = (X, U, B, w)$ of $2 - MinHom\,(Inv\,(F))$ with constraints pair $U = \langle \rho_i \rangle_{1 \leq i \leq n}$, $B = \langle \rho_{kl} \rangle_{1 \leq k \neq l \leq n}$ and its microstructure graph has a subgraph of the type $S_{2p+1}, p \geq 2$. For convenience, let us introduce $\rho_{ii} = \{(a,a) \,|a \in \rho_i\}$. Then, there is a set of pairs $\{(i_0, b_0), (i_1, b_1), \dots, (i_{2p}, b_{2p})\}$, such that for $0 \leq l \leq 2p$: $(b_l, b_{l\oplus 1}) \notin \rho_{i_l i_{l\oplus 1}}$ and $(b_l, b_{l\oplus 2}) \in \rho_{i_l i_{l\oplus 2}}$, where $i \oplus j$ denotes $i + j (\mathrm{mod}\ 2p+1)$.

From $(b_l, b_{l\oplus 2}) \in \rho_{i_l i_{l\oplus 2}}$ and the path-consistency condition $\rho_{i_l i_{l\oplus 2}} \subseteq \rho_{i_l i_{l\oplus 1}} \circ \rho_{i_{l\oplus 1} i_{l\oplus 2}}$, we see that there is $a_{l\oplus 1}$, such that $(b_l, a_{l\oplus 1}) \in \rho_{i_l i_{l\oplus 1}}$ and $(a_{l\oplus 1}, b_{l\oplus 2}) \in \rho_{i_{l\oplus 1} i_{l\oplus 2}}$.

Consider the predicate $\rho'_{l,l\oplus 1} = \rho_{i_l i_{l\oplus 1}} \cap \{a_l, b_l\} \times \{a_{l\oplus 1}, b_{l\oplus 1}\} \in Inv\,(F)$. Obviously, $\rho'_{l,l\oplus 1}$ equals to either ${}^{a_l}_{b_l}\!\overline{\times}\!{}^{a_{l\oplus 1}}_{b_{l\oplus 1}}$ or ${}^{a_l}_{b_l}\!\times\!{}^{a_{l\oplus 1}}_{b_{l\oplus 1}}$.

Let us show that if $\{a_l, b_l\} \in \overline{M}$, then $\{a_{l\oplus 1}, b_{l\oplus 1}\} \in \overline{M}$, too. Assume to the contrary that $\{a_{l\oplus 1}, b_{l\oplus 1}\} \in M$. Then, by Theorem 5.3, there is a $\phi \in F : \begin{matrix} a_{l\oplus 1} \\ \downarrow \\ b_{l\oplus 1} \end{matrix}\ \phi$, where $\phi|_{\{a_l, b_l\}}$ is a projection on the first coordinate. In this case, ϕ preserves neither ${}^{a_l}_{b_l}\!\overline{\times}\!{}^{a_{l\oplus 1}}_{b_{l\oplus 1}}$ nor ${}^{a_l}_{b_l}\!\times\!{}^{a_{l\oplus 1}}_{b_{l\oplus 1}}$, because

$$\left(\begin{array}{c} b_l \\ b_{l\oplus 1} \end{array} \right) = \left(\begin{array}{c} \phi\,(b_l, a_l) \\ \phi\,(a_{l\oplus 1}, b_{l\oplus 1}) \end{array} \right).$$

Hence, we need to consider two cases only: 1) $\forall l\ \ \{a_l, b_l\} \in M$ and 2) $\forall l\ \ \{a_l, b_l\} \in \overline{M}$. In the first case, we have $\langle (a_l, b_l), (a_{l\oplus 1}, b_{l\oplus 1}) \rangle \in P$, i.e., there is an odd cycle in T_F which contradicts that T_F is bipartite.

Now, consider the case $\forall l\ \ \{a_l, b_l\} \in \overline{M}$. By Theorem 5.4, there is a function $m \in F$, arithmetical on $\overline{M}$. If $\rho'_{l,l\oplus 1} = {}^{a_l}_{b_l}\!\overline{\times}\!{}^{a_{l\oplus 1}}_{b_{l\oplus 1}}$, then we have that

$$\left(\begin{array}{c} b_l \\ b_{l\oplus 1} \end{array} \right) = \left(\begin{array}{c} m\,(a_l, a_l, b_l) \\ m\,(b_{l\oplus 1}, a_{l\oplus 1}, a_{l\oplus 1}) \end{array} \right) \in \rho'_{l,l\oplus 1}$$

and $\rho'_{l,l\oplus 1} = {}^{a_l}_{b_l}\!\times\!{}^{a_{l\oplus 1}}_{b_{l\oplus 1}}$.

Consider the set $\{i_0, i_1, \dots, i_{2p}\}$. Suppose first that all $i_0, i_1, \dots, i_{2p}$ are distinct. Then, Theorems 5.3 and 5.4 show us that we have an arithmetical operation $m \in F$ on $\overline{M}$ and a tournament pair $\phi, \psi \in F$ on M. It is easy to see that an instance of $2 - MinHom\,(Inv\,(F))$ with constraints pair $U = \langle \rho_i \rangle_{1 \leq i \leq n}, B = \langle \rho_{kl} \rangle_{1 \leq k \neq l \leq n}$ is an odd arithmetical deadlock where $\{i_0, i_1, \dots, i_{2p}\}$ is a deadlock set. This contradicts that I is arc- and path-consistent.

The case when the elements $i_0, i_1, \ldots, i_{2p}$ are not distinct can be reduced to the previous case by the following trick: introduce a new set of variables $X' = \{(i_0, 0), (i_1, 1), \ldots, (i_{2p}, 2p)\}$ and $\rho_{(i_s,s)} = \rho_{i_s}$, where $0 \leq s \leq 2p$. If $i_m \neq i_n$, then $\rho_{(i_m,m),(i_n,n)} = \rho_{i_m,i_n}$, else $\rho_{(i_m,m),(i_n,n)} = \{(a,a)|a \in \rho_{i_m}\}$. It is easy to see that an instance with constraints pair $U = \{\rho_i\}_{i \in X'}, B = \{\rho_{kl}\}_{k \neq l \in X'}$ satisfy the conditions of Theorem 7.2 and is an odd arithmetical deadlock, where the set $\{(i_0, 0), (i_1, 1), \ldots, (i_{2p}, 2p)\}$ is a deadlock set. Therefore, we have a contradiction. ■

Proof of polynomial case of Theorem 3.4. The conditions of Theorem 3.4 coincides with the conditions of Theorem 8.1 so the microstructure graph of an arc- and path-consistent instance forbids subgraphs of the type $S_{2p+1}, p \geq 2$. By Theorem 6.8, it is perfect and, by Theorem 6.6, we see that the class F is tractable. ■

Theorems 3.3 and 3.4 give the required dichotomy for conservative algebras, which implies the dichotomy for conservative constraint languages. By Theorem 2.9, we have the following general dichotomy.

Theorem 8.2. *If $MinHom\,(\Gamma)$ is not tractable then it is NP-hard.*

9. Related work and open problems

$MinHom$ can be viewed as a problem that fits the VCSP (Valued CSP) framework by [5]. By a valued predicate of arity m over a domain D, we mean a function $p : D^m \to \mathbb{N} \cup \{\infty\}$. Informally, if Γ is a finite set of valued predicates over a finite domain D, then an instance of $VCSP(\Gamma)$ is a set of variables together with specified subsets of variables restricted by valued predicates from Γ. Any assignment to variables can be considered a solution and the measure of this solution is the sum of the values that the valued predicates take under the assignments of the specified subsets of variables. The problem is to minimize this measure. It is widely believed that a dichotomy conjecture holds for $VCSP(\Gamma)$, too.

Our dichotomy result for $MinHom$ encourages us to consider generalizations that belong to this framework.

1. Suppose we are given a constraint language Γ and a finite set of unary functions $F \subseteq \{f : D \to \mathbb{N}\}$. Let $MinHom_F(\Gamma)$ denote a minimization problem which is defined completely analogously to $MinHom(\Gamma)$ except that we are restricted to minimizing functionals of the following form: $\sum_{i=1}^{n} \sum_{f \in F} w_{if} f(x_i)$. We believe that the complexity of $MinHom_F(\Gamma)$ is determined by Γ and a certain loopless digraph $G_F = (D, \{(x,y) : \exists f \in F\ f(x) > f(y)\})$. This conjecture holds when Γ is conservative and every two vertices of G_F have an arc (of any direction) between them. Of course, a complete classification of the complexity of this problem is an open question.

2. Suppose we have a finite valued constraint language Γ, i.e. a set of valued predicates over some finite domain set. If Γ contains all unary valued predicates, we call $VCSP(\Gamma)$ a conservative $VCSP$. This name is motivated by the fact that in this case the multimorphisms (which is a generalization of polymorphisms for valued constraint languages [5]) of Γ must consist of conservative functions. Since there is a well-known dichotomy for conservative CSPs [3], we suspect that there is a dichotomy for conservative $VCSPs$.

3. $MinHom$ has (just as CSP) a homomorphism formulation. If we restrict ourselves to relational structures given by digraphs, we arrive at the following problem which we call

digraph $MinHom$: given digraphs S, H and weights $w_{ij}, i \in S, j \in H$, find a homomorphism $h : S \to H$ that minimizes the sum $\sum\limits_{s \in S} w_{sh(s)}$. Suppose we have sets of digraphs $\mathbb{G}_1, \mathbb{G}_2$. Then, $MinHom(\mathbb{G}_1, \mathbb{G}_2)$ denotes the digraph $MinHom$ problem when the first digraph is from $\mathbb{G}_1$ and the second is from $\mathbb{G}_2$. In this case, $MinHom(All, \{H\})$ coincides with $MinHom(\{H\})$ which is characterized in this paper. Another characterization based on digraph theory was announced during the preparation of the camera-ready version of this paper [16]. We believe that this approach could be fruitful for characterizing the complexity of $MinHom(\mathbb{G}, \mathbb{G})$: for example, is there a dichotomy for $MinHom(\mathbb{G}, \mathbb{G})$?

Acknowledgement

The author wishes to acknowledge fruitful discussions with Peter Jonsson and Andrei Bulatov.

References

[1] Baker K., Pixley A. Polynomial interpolation and the Chinese remainder theorem for algebraic systems. *Math. Z.*, 1975, 143, no. 2, pp. 165–174.

[2] Bodnarcuk V.G., Kalužnin L.A., Kotov N.N., Romov B.A. Galois theory for Post algebras. *Kibernetika*, Kiev, 1969, no. 3, pp. 1–10, no. 5, pp. 1–9. (in Russian)

[3] Bulatov A. Tractable conservative Constraint Satisfaction Problems. *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, 2003, pp. 321–330.

[4] Chudnovsky M., Robertson N., Seymour P., Thomas R. The strong perfect graph theorem. *Annals of Mathematics*, 2006, no. 164, pp. 51–229.

[5] Cohen D., Cooper M., Jeavons P. An algebraic characterisation of complexity for valued constraints. *Proceedings of the 12th International Conference on Principles and Practice of Contraint Programming*, 2006, pp. 107–121.

[6] Feder T., Vardi M. Y. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 1999, no. 28(1), pp. 57–104.

[7] Geiger D. Closed Systems of Functions and Predicates. *Pacific Journal of Mathematics*, 1968, no. 27, pp. 95–100.

[8] Gupta A., Hell P., Karimi M., Rafiey A. Minimum cost homomorphisms to reflexive digraphs. *LATIN*, 2008.

[9] Gutin G., Hell P., Rafiey A., Yeo A. A dichotomy for minimum cost graph homomorphisms. *European Journal of Combinatorics*, 2008, Volume 29, Issue 4, pp. 900–911.

[10] Gutin G., Hell P., Rafiey A., Yeo A. Minimum cost and list homomorphisms to semicomplete digraphs. *Discrete Appl. Math.*, 2006, Volume 154, pp. 890–897.

[11] Gutin G., Rafiey A., Yeo A., Tso M. Level of repair analysis and minimum cost homomorphisms of graphs. *Discrete Applied Mathematics*, no. 154(6), pp. 881–889.

[12] Gutin G., Rafiey A., Yeo A. Minimum Cost Homomorphism Dichotomy for Oriented Cycles. *Proceedings of AAIM'08*, Lecture Notes in Computer Science, 2008, 5034, pp. 224–234.

[13] Jeavons P. On the Algebraic Structure of Combinatorial Problems. *Theoretical Computer Science*, 1998, no. 200, 1–2, pp. 185–204.

[14] Jégou P. Decomposition of domains based on the micro-structure of finite constraint satisfaction problems. Proceedings of the 11th National Conference on Artificial Intelligence, 1993, pp. 731–736.

[15] Post E. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, Princeton University Press, 1941, no. 5.

[16] Rafiey A., Hell P. Duality for Min-Max Orderings and Dichotomy for Min Cost Homomorphisms. http://arxiv.org/abs/0907.3016v1

[17] Schaefer T.J. The complexity of satisfiability problems. *Proc 10th ACM Symposium on Theory of Computing (STOC)*, 1978, pp. 216–226.

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 669-680
www.stacs-conf.org

ALTERNATION-TRADING PROOFS, LINEAR PROGRAMMING, AND LOWER BOUNDS (EXTENDED ABSTRACT)

RYAN WILLIAMS[1]

[1] IBM Almaden Research Center
650 Harry Road, San Jose, CA, USA 95120
E-mail address: ryanwill@us.ibm.com
URL: http://www.cs.cmu.edu/ ryanw/

Abstract. A fertile area of recent research has demonstrated concrete polynomial time lower bounds for solving natural hard problems on restricted computational models. Among these problems are Satisfiability, Vertex Cover, Hamilton Path, MOD_6-SAT, Majority-of-Majority-SAT, and Tautologies, to name a few. The proofs of these lower bounds follow a certain proof-by-contradiction strategy that we call *alternation-trading*. An important open problem is to determine how powerful such proofs can possibly be.

We propose a methodology for studying these proofs that makes them amenable to both formal analysis and automated theorem proving. We prove that the search for better lower bounds can often be turned into a problem of solving a large series of linear programming instances. Implementing a small-scale theorem prover based on this result, we extract new human-readable time lower bounds for several problems. This framework can also be used to prove concrete limitations on the current techniques.

1. Introduction

Many known lower bounds for natural problems follow a type of algorithmic argument that we call a *resource-trading proof.* Such a proof assumes that a hard problem *can* be solved by a "good" algorithm, and tries to derive a contradiction by combining two essential components. One is a *speedup lemma*, which simulates all good algorithms super-efficiently on some "interesting" computational model, trading time for some resource. The second component is a *slowdown lemma*, which uses the assumed good algorithm for the hard problem to simulate computations from the "interesting" model by good algorithms, thereby trading the "interesting" resource for more time. Clever combinations of speedup and slowdown lemmas are used to contradict a known result, in particular some complexity

1998 ACM Subject Classification: F.2.3, I.2.3.

Key words and phrases: time-space tradeoffs, lower bounds, alternation, linear programming.

This material is based on work supported in part by NSF grant CCR-0122581 while the author was a student at Carnegie Mellon University, and NSF grant CCF-0832797 while the author was a member of the Institute for Advanced Study.

SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2494

hierarchy theorem. That is, by assuming a "good" algorithm for a hard problem, we derive something like $\mathsf{TIME}[n^2] \subseteq \mathsf{TIME}[n]$, a contradiction.

As an example, one can prove a time-space tradeoff for satisfiability (SAT) as follows. Assume SAT has an algorithm running in n^c time and $\mathrm{poly}(\log n)$ space, for some $c > 1$. One speedup lemma is that computations running in n^a time and $\mathrm{poly}(\log n)$ space can be simulated by an *alternating machine* that switches from co-nondeterministic mode to nondeterministic mode once (i.e., a Π_2 machine), and runs in $n^{a/2+o(1)}$ time. This speedup lemma *trades time for alternations*. The relevant slowdown lemma is: if SAT has an n^c time, $\mathrm{poly}(\log n)$ space algorithm, then (by a strengthening of the Cook-Levin theorem) every language in $\mathsf{NTIME}[t]$ has $t^{c+o(1)}$ time, $\mathrm{poly}(\log t)$ space algorithms. Consequently, an alternating machine running in t time and making $k-1$ alternations has $t^{c^k+o(1)}$ time, $\mathrm{poly}(\log t)$ space algorithms. Combining these speedup and slowdown lemmas, we derive

$$\Sigma_2\mathsf{TIME}[t] \subseteq \mathsf{DTISP}[t^{c^2+o(1)}, \mathrm{poly}(\log t)] \subseteq \Pi_2\mathsf{TIME}[t^{c^2/2}],$$

where the first inclusion holds by slowdown and the second holds by speedup. Now observe that the alternating time hierarchy is contradicted when $c^2 < 2$. This proof is the $n^{\sqrt{2}-\varepsilon}$ time lower bound of Lipton and Viglas [LV99].

Some of the best known separations in complexity theory use resource-trading proofs. Hopcroft, Paul, and Valiant [HPV77] showed that $\mathsf{SPACE}[n] \not\subseteq \mathsf{DTIME}[o(n \log n)]$ for multitape Turing machines, by proving the "speedup lemma" that $\mathsf{DTIME}[t] \subseteq \mathsf{SPACE}[t/\log t]$ and invoking diagonalization. Their result was later extended to general models [PR81, HLMW86]. Paul, Pippenger, Szemeredi, and Trotter [PPST83] proved that $\mathsf{NTIME}[n] \neq \mathsf{DTIME}[n]$ for multitape Turing machines. The key component in the proof is the "speedup lemma" $\mathsf{DTIME}[t] \subseteq \Sigma_4\mathsf{TIME}[t/\log^* t]$ for multitape TMs. Despite their age, the above separations still constitute the best known progress on P vs PSPACE and P vs NP, respectively.

In more recent years, resource-trading proofs have established time-space lower bounds for NP-complete problems and problems higher in the polynomial hierarchy [Kan84, For97, LV99, FvM00, FLvMV05, Wil06, Wil08]. For instance, the best known time lower bound for solving SAT with $n^{o(1)}$-space algorithms is $n^{2\cos(\pi/7)-o(1)} \geq n^{1.801}$, obtained with a resource-trading proof [Wil08]. (Note if one could improve the 1.801 exponent to arbitrary constants, one would separate $\mathsf{LOGSPACE}$ from NP.) For nondeterministic algorithms using $n^{o(1)}$ space, the best known time lower bound for solving the coNP-complete TAUTOLOGY problem was $n^{\sqrt{2}-o(1)}$ for several years [FvM00]. Certain time-space lower bounds for probabilistic and quantum computations also follow the resource-trading paradigm [AKRRV01, DvM06, Vio09, vMW07]. Resource-trading proofs are also abound in the multidimensional "hybrid" Turing machine model, which has read-only random access to its input and an $n^{o(1)}$ read-write store, as well as read-write two-way access to a d-dimensional tape for some $d \geq 1$. This is the most powerful (and physically realistic) model known where we still know nontrivial time lower bounds for problems such as SAT. Multidimensional TMs have a long history; e.g., [Lou80, PR81, Kan83, MS87, vMR05, Wil06] proved lower bounds for them. (For a more complete literature review, please see the full version of the paper.)

1.1. Main Results

We introduce a methodology for reasoning about resource-trading proofs that is also practically implementable for finding short proofs. Informally, the "hard work" in these proofs can be replaced by solving a series of linear programming problems. This perspective not only aids us practically in the search for new lower bounds, but also allows us to show non-trivial limitations on what can be proved.

This methodology is applied to several lower bound problems. In all cases considered here, the resource being "traded" is alternations, so we call the proofs *alternation-trading.*

Deterministic Time-Space Lower Bounds. Aided by results of a computer program, we show that any SAT algorithm running in $t(n)$ time and $s(n)$ space satisfies $t \cdot s \geq \Omega(n^{2\cos(\pi/7)-o(1)})$. Previously, the best known result was $t \cdot s \geq \Omega(n^{1.573})$ [FLvMV05]. It has been conjectured that the current framework sufficed to prove a $n^{2-o(1)}$ time lower bound for SAT, against algorithms using $n^{o(1)}$ space. We prove that it is not possible to obtain n^2 with the framework, formalizing a conjecture of [FLvMV05].[*] A computer search over proofs of short length suggests that the best known $n^{2\cos(\pi/7)-o(1)}$ lower bound [Wil08] is already optimal for the framework. We also prove lower bounds on QBF_k (quantified Boolean formulas with at most k quantifier blocks), showing that the problem requires $\Omega(n^{k+1-\delta_k})$ time for $n^{o(1)}$ space algorithms, where $\delta_k < 0.2$ and $\lim_{k\to\infty} \delta_k = 0$.[†]

Nondeterministic Time-Space Lower Bounds. Adapting our ideas to proving lower bounds for TAUTOLOGIES, a computer program found a very short proof improving upon Fortnow and Van Melkebeek's lower bound. Longer proofs suggested an interesting pattern. Joint work with Diehl and Van Melkebeek on this observation resulted in an $n^{4^{1/3}-o(1)} \geq n^{1.587}$ time lower bound [DvMW09]. Computer search suggests that this lower bound is best possible for the framework. We prove that it is not possible to obtain an n^{ϕ} time lower bound, where $\phi = 1.618\ldots$ is the golden ratio. This is surprising since we have known for some time that an n^{ϕ} lower bound *is* provable for *deterministic* algorithms [FvM00].

Multidimensional Turing Machine Lower Bounds. Here our method uncovers peculiar behavior in the best lower bound proofs, regardless of the dimension. Studying computer search results, we extract an $\Omega(n^{r_d})$ time lower bound for the d-dimensional case, where $r_d \geq 1$ is the root of a particular quintic $p_d(x)$ with coefficients depending on d. For example, $r_1 \approx 1.3009$, $r_2 \approx 1.1887$, and $r_3 \approx 1.1372$. Again, our search suggests this is best possible, and we can prove it is not possible to improve the bound for d-dimensional TMs to $n^{1+1/(d+1)}$ with the current tools.

These limitations also hold for other NP and coNP-hard problems; the only property required is that all languages in $\mathsf{NTIME}[n]$ (respectively, $\mathsf{coNTIME}[n]$) have sufficiently efficient reductions to the problem. Also our linear programming approach is not limited to the above, and can be applied to the league of lower bounds discussed in Van Melkebeek's surveys [vM04, vM07].

1.2. Some Remarks on the Reduction to Linear Programming

The key to our formulation is to separate the *discrete choices* in an alternation-trading proof from the *real-valued choices.* The discrete choices consist of the sequence of lemmas to apply in each step, and what sort of hierarchy theorem to use in the contradiction. We present several simplifications that greatly reduce the number of discrete choices, without loss of generality. The real-valued choices are the running time exponents that arise from

[*] That is, we formalize the statement: "...some complexity theorists feel that improving the golden ratio exponent beyond 2 would require a breakthrough" in Section 8 of [FLvMV05].

[†] Note the QBF_k results appeared in the author's PhD thesis in 2007 but have been unpublished to date.

the choices of time bounds and rule applications. We prove that once the discrete choices are made, the remaining real-valued problem can be expressed as an instance of linear programming. This makes it possible to search for new proofs via computer, and it also gives us a formal handle on the limitations of these proofs.

One cannot easily search over all possible proofs, as the number of discrete choices is still about $2^n/n^{3/2}$ for proofs of n lines (proportional to the nth Catalan number). Nevertheless it is still feasible to try all 24+ line proofs. These proof searches reveal patterns, indicating that certain strategies will be most successful in proving lower bounds; in each case we study, the resulting strategies differ. Following the strategies, we establish new lower bound proofs. The patterns also suggest how to show limitations on the proof systems.

Note: *Due to space limitations, we can only describe how our methods apply to SAT time-space lower bounds. Please see the full version of the paper for proofs and more details.*

2. Preliminaries

We assume familiarity with Complexity Theory, especially the notion of alternationWe use big-Ω notation in the infinitely often sense, so statements like "SAT is not in $O(n^c)$ time" are equivalent to "SAT requires $\Omega(n^c)$ time." All functions are assumed constructible within the appropriate bounds. Our default computational model is the random access machine, broadly construed: particular variants do not affect the results. $\mathsf{DTISP}[t(n), s(n)]$ is the class of languages accepted by a RAM running in $t(n)$ time and $s(n)$ space, simultaneously. For convenience, we set $\mathsf{DTS}[t(n)] := \mathsf{DTISP}[t(n)^{1+o(1)}, n^{o(1)}]$ to omit negligible $o(1)$ factors.

In order to properly formalize alternation-trading proofs, we introduce notation for alternating complexity classes that include *input constraints* between alternations. Let us start with an example of the notation, then give a general definition. Define $(\exists\ f(n))^b\mathsf{DTS}[n^a]$ to be the class of languages recognized by a machine which, on an input x of length n, writes a $f(n)^{1+o(1)}$ bit string y nondeterministically, copies at most $n^{b+o(1)}$ bits z from the pair $\langle x, y\rangle$ (in $O(n^{b+o(1)})$ time), then feeds z as input to a machine M running in $n^{a+o(1)}$ time and $n^{o(1)}$ space. Note the runtime of M is measured with respect to the initial input length n, not the latter input length $n^{b+o(1)}$ of z.

We generalize this definition as follows. Let $\mathcal{C}$ be a complexity class. For $i = 1, \ldots, k$, let $Q_i \in \{\exists, \forall\}$ and $a_i, b_i \geq 0$. Define

$$(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2}) \cdots^{b_k} (Q_k\ n^{a_k})^{b_{k+1}}\mathcal{C}$$

to be the class of languages recognized by a machine M that, on input x of length n, has the following general behavior on input x:

```
Set $z_0 := x$.
For $i = 1, \ldots, k$,
    If $Q_i = \exists$, switch to existential mode.
    If $Q_i = \forall$, switch to universal mode.
    Guess an $n^{a_i+o(1)}$ bit string $y$ (universally or existentially).
    Copy at most $n^{b_{i+1}+o(1)}$ bits $z_i$ from the pair $\langle z_{i-1}, y\rangle$.
End for
Run a machine recognizing a language in class $\mathcal{C}$ on the input $z_k$.
```

When an input constraint b_i is unspecified, its default value is $\max\{a_i, 1\}$. We say that the existential and universal modes of an alternating computation are *quantifier blocks*, to reflect the complexity class notation. It is crucial to observe that the time bound in the ith quantifier block is measured with respect to n, the input to the *first quantifier block*.

Notice that by simple properties of nondeterminism and conondeterminism, we can combine adjacent quantifier blocks that are of the same type, e.g., $(\exists n^a)^a(\exists n^b)^b\mathsf{DTS}[n^c] = (\exists n^{\max\{a,b\}})^b\mathsf{DTS}[n^c]$. This useful property is exploited in alternation-trading proofs.

2.1. A Short Introduction to Alternation-Trading Proofs

Here we give a brief overview of the tools used in alternation-trading proofs. In this extended abstract we focus on deterministic time lower bounds for satisfiability for algorithms using $n^{o(1)}$ workspace; the other lower bound problems use similar tools.

It is known that satisfiability of Boolean formulas in conjunctive normal form (SAT) is a complete problem under tight reductions for a small nondeterministic complexity class. The class NQL, called *nondeterministic quasilinear time*, is defined as

$$\mathsf{NQL} := \bigcup_{c \geq 0} \mathsf{NTIME}[n \cdot (\log n)^c] = \mathsf{NTIME}[n \cdot \mathit{poly}(\log n)].$$

Theorem 2.1 ([Coo88, Sch78, Tou01, FLvMV05]). *SAT is NQL-complete under quasilinear time $O(\log n)$ space reductions, for both multitape and random access machine models. Moreover, each bit of the reduction can be computed in $O(\mathit{poly}(\log n))$ time and $O(\log n)$ space in both machine models.*[‡]

Let $\mathcal{C}[t(n)]$ represent a time $t(n)$ complexity class under one of the three models:

- deterministic RAM using time t and $t^{o(1)}$ space,
- co-nondeterministic RAM using time t and $t^{o(1)}$ space,
- d-dimensional Turing machine using time t.

Theorem 2.1 implies that if $\mathsf{NTIME}[n] \not\subseteq \mathcal{C}[t]$, then SAT $\notin \mathcal{C}[t/\text{poly}(\log t)]$.

Corollary 2.2. *If $\mathsf{NTIME}[n] \not\subseteq \mathcal{C}[t(n)]$, then SAT $\notin \mathcal{C}[t(n)/\log^k t(n)]$ for some $k > 0$.*

Hence we wish to prove $\mathsf{NTIME}[n] \not\subseteq \mathcal{C}[n^c]$ for large $c > 1$. To prove time-space lower bounds, we work with $\mathcal{C}[n^c] = \mathsf{DTS}[n^c] = \mathsf{DTISP}[n^c, n^{o(1)}]$. Van Melkebeek and Raz [vMR05] observed that a similar corollary holds for any problem Π such that SAT reduces to Π under highly efficient reductions, *e.g.* Vertex Cover, Hamilton Path, 3-SAT, and Max-2-Sat. Therefore similar time lower bounds hold for these problems as well.

Speedups, Slowdowns, and Contradictions. Now that our goal is to prove $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTS}[n^c]$, how can we do this? In an alternation-trading proof, we attempt to establish a contradiction from assuming $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, by applying two lemmas which complement one another. A *speedup lemma* takes a $\mathsf{DTS}[t]$ class and places it in an alternating class with runtime $o(t)$. A *slowdown lemma* takes an alternating class with runtime t and places it in a class with one less alternation and runtime $O(t^c)$. The Speedup Lemma dates back to Nepomnjascii [Nep70] and Kannan [Kan84].

[‡] In the multitape Turing machine model we assume that the tape heads are already oriented on the appropriate cells, otherwise it may take linear time to find the appropriate cells on a tape.

Lemma 2.3 (Speedup Lemma)**.** *Let $a \geq 1$, $e \geq 0$ and $0 \leq x \leq a$. Then*

$$\mathsf{DTISP}[n^a, n^e] \subseteq (Q_1\ n^{x+e})^{\max\{1,x+e\}}(Q_2\ \log n)^{\max\{1,e\}}\mathsf{DTISP}[n^{a-x}, n^e],$$

for $Q_i \in \{\exists, \forall\}$ where $Q_1 \neq Q_2$. In particular,

$$\mathsf{DTS}[n^a] \subseteq (Q_1\ n^x)^{\max\{1,x\}}(Q_2\ \log n)^1\mathsf{DTS}[n^{a-x}].$$

Proof. Let M use n^a time and n^e space. Let y be an input of length n. A complete description (i.e. *configuration*) of $M(y)$ at any step can be described in $O(n^e + \log n)$ space. To simulate M in $(\exists\ n^{x+e})^{\max\{1,x+e\}}(\forall\ \log n)^{\max\{1,e\}}\mathsf{DTISP}[n^{a-x}, n^e]$, the algorithm $N(y)$ existentially guesses a sequence of configurations $C_1, \ldots, C_{n^x}$ of $M(x)$. Then $N(y)$ appends the initial configuration C_0 of $M(y)$ to the beginning of the sequence, and an accepting configuration C_{n^x+1} to the end. $N(y)$ universally guesses a $i \in \{0, \ldots, n^x\}$, erases all configurations except C_i and C_{i+1}, then simulates $M(y)$ starting from C_i, accepting if and only if C_{i+1} is reached within n^{a-x} steps. It is easy to see the simulation is correct. The input constraints on the quantifier blocks are satisfied since after the universal guess, the input is only y, C_i, and C_{i+1}, which is of size $n + 2n^{e+o(1)} \leq n^{\max\{1,e\}+o(1)}$. ■

Observe in the above alternating simulation, the input to the final DTISP computation is linear in $n + n^e$, *regardless of the choice of x*. This is a subtle property that is exploited heavily in alternation-trading proofs. The Slowdown Lemma is the following simple result:

Lemma 2.4 (Slowdown Lemma)**.** *Let $a \geq 1$, $e \geq 0$, $a' \geq 0$, and $b \geq 1$. If $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^e]$, then for both $Q \in \{\exists, \forall\}$,*

$$(Q\ n^{a'})^b\mathsf{DTIME}[n^a] \subseteq \mathsf{DTISP}[n^{c\cdot\max\{a,a',b\}}, n^{e\cdot\max\{a,a',b\}}].$$

In particular, if $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, then

$$(Q\ n^{a'})^b\mathsf{DTIME}[n^a] \subseteq \mathsf{DTS}[n^{c\cdot\max\{a,a',b\}}].$$

Proof. Let L be a problem in $(Q\ n^{a'})^b\mathsf{DTIME}[n^a]$, and let A be an algorithm recognizing L. On an input x of length n, A guesses a string y of length $n^{a'+o(1)}$ and feeds an $n^{b+o(1)}$ bit string z to $A'(z)$, where A' is a deterministic algorithm that runs in n^a time. Since $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^e]$ and DTISP is closed under complement, by padding we have $\mathsf{NTIME}[p(n)] \cup \mathsf{coNTIME}[p(n)] \subseteq \mathsf{DTISP}[p(n)^c, p(n)^e]$ for polynomials $p(n) \geq n$. Therefore A can be simulated with a deterministic algorithm B. Since the runtime of A is $n^{a'+o(1)} + n^{b+o(1)} + n^a$, the runtime of B is $n^{c\cdot\max\{a,a',b\}+o(1)}$ and the space usage is similar. ■

The final component of an alternation-trading proof is a time hierarchy theorem, the most general of which is the following, provable by a simple diagonalization.

Theorem 2.5 (Alternating Time Hierarchy)**.** *For $k \geq 0$, for all $Q_i \in \{\exists, \forall\}$, $1 \leq a'_i < a_i$, and $1 \leq b'_i \leq b_i$,*

$$(Q_1\ n^{a_1})^{b_2} \cdots^{b_k} (Q_k\ n^{a_k})^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}] \nsubseteq (R_1\ n^{a'_1})^{b'_2} \cdots^{b'_k} (R_k\ n^{a'_k})^{b'_{k+1}}\mathsf{DTS}[n^{a'_{k+1}}],$$

where $R_i \in \{\exists, \forall\}$ and $R_i \neq Q_i$ for all $i = 2, \ldots, k+1$.

Two Examples. Let us give a couple of examples of alternation-trading proofs. To simplify the presentation we do not specify the input constraints to quantifiers in the below.

(1) In FOCS'99, Lipton and Viglas proved that SAT cannot be solved by algorithms running in $n^{\sqrt{2}-\varepsilon}$ time and $n^{o(1)}$ space, for all $\varepsilon > 0$. By Theorem 2.1, if SAT is in $n^{\sqrt{2}-\varepsilon}$ time and $n^{o(1)}$ space then $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ with $c^2 < 2$. We have

$$\begin{aligned}(\exists\ n^{2/c^2})(\forall\ n^{2/c^2})\mathsf{DTS}[n^{2/c^2}] &\subseteq (\exists\ n^{2/c^2})\mathsf{DTS}[n^{2/c}] && \text{(Slowdown Lemma)}\\ &\subseteq \mathsf{DTS}[n^2] && \text{(Slowdown Lemma)}\\ &\subseteq (\forall\ n)(\exists\ \log n)\mathsf{DTS}[n]. && \text{(Speedup Lemma, with } x=1)\end{aligned}$$

But $(\exists\ n^{2/c^2})(\forall\ n^{2/c^2})\mathsf{DTS}[n^{2/c^2}] \subseteq (\forall\ n)(\exists\ \log n)\mathsf{DTS}[n]$ contradicts Theorem 2.5. In fact, one can show that if $c^2 = 2$, we still have a contradiction with $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, so the ε can be removed from the previous statement and state that SAT cannot be solved in $n^{\sqrt{2}}$ time and $n^{o(1)}$ exactly.[§]

(2) Improving on the previous example, one can show SAT $\notin \mathsf{DTS}[n^{1.6004}]$. If $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ and $\sqrt{2} \leq c < 2$, then applying the Speedup and Slowdown Lemmas one can derive:

$$\begin{aligned}\mathsf{DTS}[n^{c^2/2+2}] &\subseteq (\exists\ n^{c^2/2})(\forall\ \log n)\mathsf{DTS}[n^2] && \text{(Speedup)}\\ &\subseteq (\exists\ n^{c^2/2})(\forall\ \log n)(\forall\ n)(\exists\ \log n)\mathsf{DTS}[n] && \text{(Speedup)}\\ &= (\exists\ n^{c^2/2})(\forall\ n)(\exists\ \log n)\mathsf{DTS}[n] && \text{(Combining } \forall \text{ Quantifiers)}\\ &\subseteq (\exists\ n^{c^2/2})(\forall\ n)\mathsf{DTS}[n^c] && \text{(Slowdown)}\\ &\subseteq (\exists\ n^{c^2/2})\mathsf{DTS}[n^{c^2}] && \text{(Slowdown)}\\ &\subseteq (\exists\ n^{c^2/2})(\exists\ n^{c^2/2})(\forall\ \log n)\mathsf{DTS}[n^{c^2/2}] && \text{(Speedup)}\\ &= (\exists\ n^{c^2/2})(\forall\ \log n)\mathsf{DTS}[n^{c^2/2}] && \text{(Combining } \exists \text{ Quantifiers)}\\ &\subseteq (\exists\ n^{c^2/2})\mathsf{DTS}[n^{c^3/2}] && \text{(Slowdown)}\\ &\subseteq \mathsf{DTS}[n^{c^4/2}] && \text{(Slowdown)}\end{aligned}$$

When $c^2/2 + 2 > c^4/2$ (which happens if $c < 1.6004$), we have $\mathsf{DTS}[n^a] \subseteq \mathsf{DTS}[n^{a'}]$ for some $a > a'$. One can show by a translation argument (similar to the footnote) that either $\mathsf{DTS}[n^a] \nsubseteq \mathsf{DTS}[n^{a'}]$ or $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTS}[n^c]$, concluding the proof.

Example (2) was discovered by a computer program. By "discovered", we mean that the program applied speedups and slowdowns in precisely the same way, having only minimum knowledge of the lemmas. Furthermore, the program verified that above is the *best possible* alternation-trading proof that applies the Speedup and Slowdown Lemmas at most 7 times. A more formal definition of "alternation-trading proof" is given in the next section.

3. Formalizing Alternation-Trading Proofs

We formalize alternation-trading proofs of lower bounds on DTS classes as follows:[¶]

Definition 3.1. Let $c > 1$. An *alternation-trading proof for c* is a list of complexity classes of the form:

$$(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{a_k})^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}], \tag{3.1}$$

[§] Suppose $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ and $\mathsf{\Sigma_2TIME}[n] \subseteq \mathsf{\Pi_2TIME}[n^{1+o(1)}]$. The first assumption, along with the Speedup and Slowdown Lemmas, implies that for every k there's a K satisfying $\mathsf{\Sigma_2TIME}[n^k] \subseteq \mathsf{NTIME}[n^{kc}] \subseteq \mathsf{\Sigma}_K\mathsf{TIME}[n]$. But the second assumption implies that $\mathsf{\Sigma}_K\mathsf{TIME}[n] = \mathsf{\Sigma_2TIME}[n^{1+o(1)}]$. Hence $\mathsf{\Sigma_2TIME}[n^k] \subseteq \mathsf{\Sigma_2TIME}[n^{1+o(1)}]$, which contradicts the time hierarchy for $\mathsf{\Sigma_2TIME}$.

[¶] This formalization has *implicitly* appeared in several prior works, but not to the degree that we investigate in this paper.

where $k \geq 0$, $Q_i \in \{\exists, \forall\}$, $Q_i \neq Q_{i+1}$, $a_i > 0$, and $b_i \geq 1$, for all i. (When $k = 0$, the class is deterministic.) The items of the list are called *lines of the proof.* Each line is obtained from the previous line by applying either a *speedup rule* or a *slowdown rule.* More precisely, if the ith line is

$$(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{a_k})^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}],$$

then the $(i+1)$st line has one of four possible forms:

Speedup Rule 0: For $k = 0$ and any $x \in (0, a_1)$, $(Q_0\ n^x)^{\max\{x,1\}}(Q_1\ n^0)^1\mathsf{DTS}[n^{a_1-x}]$.[‖]

Speedup Rule 1: For $k > 0$ and any $x \in (0, a_{k+1})$,

$$(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{\max\{a_k,x\}})^{\max\{x,b_{k+1}\}}(Q_{k+1}\ n^0)^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}-x}].$$

Speedup Rule 2: For $k > 0$ and any $x \in (0, a_{k+1})$,

$$(Q_1\ n^{a_1})^{b_2}\cdots^{b_k}(Q_k\ n^{a_k})^{b_{k+1}}(Q_{k+1}\ n^x)^{\max\{x,b_{k+1}\}}(Q_{k+2}\ n^0)^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}-x}].$$

Slowdown Rule: For $k > 0$,

$$(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_{k-1}}(Q_{k-1}\ n^{a_{k-1}})^{b_k}\mathsf{DTS}[n^{c\cdot\max\{a_{k+1},a_k,b_k,b_{k+1}\}}].$$

An alternation-trading proof *shows* ($\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c] \implies A_1 \subseteq A_2$) if its first line is A_1 and its last line is A_2.

The above definition comes directly from the Speedup Lemma (Lemma 2.3) and Slowdown Lemma (Lemma 2.4). The rules are easily verified to be syntactic formulations of the corresponding lemmas. For instance, Speedup Rule 1 holds, as

$$\begin{aligned}
&(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{a_k})^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}]\\
&\subseteq (Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{a_k})^{b_{k+1}}(Q_k\ n^{x})^{\max\{b_{k+1},x\}}(Q_{k+1}\ n^0)^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}]\\
&\subseteq (Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{\max\{a_k,x\}})^{\max\{b_{k+1},x\}}(Q_{k+1}\ n^0)^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}].
\end{aligned}$$

Rule 2 is akin to Rule 1, except that it uses opposite quantifiers in its invocation of the Speedup Lemma. The Slowdown Rule works analogously to Lemma 2.4. It follows that alternation-trading proofs are sound.

Note Speedup Rules 0 and 2 add two quantifier blocks, Speedup Rule 1 adds one quantifier, and all three rules introduce a parameter x. By considering "normal form" proofs (defined in the following paragraphs), we can prove that Rule 2 can always be replaced by applications of Rule 1. (A proof is in the full version of the paper.) For this reason we just refer to *the Speedup Rule*, depending on which of Rule 0 or Rule 1 applies.

Define a class of the form (3.1) to be *simple.* Define classes A_1 and A_2 to be *complementary* if A_1 is the class of complements of languages in A_2. Every known (model-independent) time-space lower bound for SAT shows "$\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ implies $A_1 \subseteq A_2$", for some complementary simple classes A_1 and A_2, contradicting a time hierarchy (cf. Theorem 2.5). A similar claim holds for nondeterministic time-space lower bounds against tautologies (which prove "$\mathsf{NTIME}[n] \subseteq \mathsf{coNTS}[n^c]$ implies $A_1 \subseteq A_2$"), for d-dimensional TM lower bounds (which prove "$\mathsf{NTIME}[n] \subseteq \mathsf{DTIME}_d[n^c]$ implies $A_1 \subseteq A_2$"), and other problems.

Normal Form. It will be very convenient to introduce a *normal form* for alternation-trading proofs. We show that any lower bound provable with complementary simple classes can also be established with a normal form proof. This greatly reduces the degrees of freedom in a proof, as we no longer need to worry about *which* time hierarchy to contradict.

[‖] Please note that the $(k+1)$th quantifier is n^0 in order to account for the $O(\log n)$ size of the quantifier.

Definition 3.2. Let $c \geq 1$. An alternation-trading proof for c is in *normal form* if (a) the first and last lines are $\mathsf{DTS}[n^a]$ and $\mathsf{DTS}[n^{a'}]$ respectively, for some $a \geq a'$, and (b) no other lines are DTS classes.

We show that a normal form proof for c implies that $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTS}[n^c]$.

Lemma 3.3. *Let $c \geq 1$. If there is an alternation-trading proof for c in normal form having at least two lines, then $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTS}[n^c]$.*

Theorem 3.4. *Let A_1 and A_2 be complementary. If there is an alternation-trading proof P for c that shows $(\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c] \Longrightarrow A_1 \subseteq A_2)$, then there is a normal form proof for c, of length at most that of P.*

Proofs of Lemma 3.3 and Theorem 3.4 are in the full version. The upshot of these results is that we may focus our proof search on normal form proofs. For the remainder of this section, we assume all alternation-trading proofs are in normal form.

Proof Annotations. Different lower bound proofs can result in quite different sequences of speedups and slowdowns. A *proof annotation* represents such a sequence.

Definition 3.5. A *proof annotation* for an alternation-trading proof of ℓ lines is the $(\ell-1)$-bit vector A where for all $i = 1, \ldots, \ell-1$, $A[i] = 1$ (respectively, $A[i] = 0$) if the ith line applies a Speedup Rule (respectively, a Slowdown Rule).

An $(\ell-1)$-bit proof annotation corresponds to a "strategy" for an ℓ-line proof. For a normal form proof of ℓ lines, it is not hard to show that its annotation A must have $A[1] = 1$, $A[\ell-2] = 0$, and $A[\ell-1] = 0$.

Note that an annotation *does not* determine a proof entirely, as other parameters need optimizing. (The problem of optimizing them is tackled in the next section.) To illustrate the annotation concept, we give four examples.

- The $n^{\sqrt{2}}$ lower bound of Lipton and Viglas has the annotation $[1, 0, 0]$.
- The $n^{1.6004}$ bound from Section 2.1 corresponds to $[1, 1, 0, 0, 1, 0, 0]$.
- The n^{ϕ} bound of Fortnow and Van Melkebeek [FvM00] is an inductive proof, corresponding to an infinite sequence of annotations. In normal form, the sequence is $[1, 0, 0], [1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0, 0], \ldots$
- The $n^{2\cos(\pi/7)}$ bound [Wil08] has two inductive stages. Let $A = 1, 0, 1, 0, \ldots, 1, 0, 0$, where the '$\ldots$' contain any number of repetitions. The sequence is
$$[A], [1, A, A], [1, 1, A, A, A], [1, 1, 1, A, A, A, A], \ldots$$
That is, the proof performs many speedups, then a sequence of many slowdown-speedup alternations, then two consecutive slowdowns, repeating this until all the quantifiers have been removed.

3.1. Translation To Linear Programming

Given a (normal form) proof annotation, how can we determine the best proof possible with it? We need to optimally set the runtimes of the first and last DTS classes in the proof, as well as the x_i parameters that arise from each application of a Speedup Rule. It turns out that an annotation A and $c > 1$ can be reduced to a polynomial size linear program that is feasible if and only if there is an alternation-trading proof of $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTS}[n^c]$

with annotation A. More precisely, the problem of optimizing parameters can be viewed as an arithmetic circuit evaluation, where the circuit has max gates, addition gates, and input gates that may multiply their input by c. Such circuits can be evaluated using a linear program that minimizes the sum of the gate values (cf. [Der72]).

Let A be an annotation of $\ell - 1$ bits, and let m be the maximum number of quantifier blocks in any line of A (note m is easily computed in linear time). The target LP has variables $a_{i,j}$, $b_{i,j}$, and x_i, for all $i = 0, \ldots, \ell - 1$ and $j = 1, \ldots, m$. The variables $a_{i,j}$ represent the runtime exponent of the jth quantifier block in the class on the ith line, $b_{i,j}$ is the input exponent to the jth quantifier block of the class on the ith line, and for all lines i that use a Speedup Rule, x_i is the choice of x in the Speedup Rule. For example:

- If the kth line of a proof is $\mathsf{DTS}[n^a]$, the corresponding constraints are
$$a_{k,1} = a,\ b_{k,1} = 1,\ (\forall k > 0)\ a_{k,i} = b_{k,i} = 0.$$
- If the kth line of a proof is $(\exists\ n^{a'})^b\mathsf{DTS}[n^a]$, then the constraints are
$$a_{k,0} = a,\ b_{k,1} = b,\ a_{k,1} = a',\ b_{k,1} = 1,\ (\forall k > 1)\ a_{k,i} = b_{k,i} = 0.$$

The objective is to minimize $\sum_{i,j}(a_{i,j} + b_{i,j}) + \sum_i x_i$. The LP constraints depend on the lines of the annotation, as follows.

Initial Constraints. For the 0th and $(\ell - 1)$th lines we have $a_{0,1} \geq a_{\ell-1,1}$, and

$a_{0,1} \geq 1,\ b_{0,1} = 1,\ (\forall\ k > 1)\ a_{0,k} = b_{0,k} = 0,$ and $a_{\ell,1} \geq 1,\ b_{\ell,1} = 1,\ (\forall k > 1)\ a_{\ell,k} = b_{\ell,k} = 0,$

representing $\mathsf{DTS}[n^{a_{0,1}}]$ and $\mathsf{DTS}[n^{a_{\ell-1,0}}]$, respectively. The 1st line of a proof always applies Speedup Rule 1, having the form $(Q_1 n^x)^{\max\{x,1\}}(Q_2\ n^0)^1\mathsf{DTS}[n^{a-x}]$. So the constraints for the 1st line are:

$$a_{1,1} = a_{0,1} - x_1,\ b_{1,1} = 1,\ a_{1,2} = 0,\ b_{1,2} \geq x_1,\ b_{1,2} \geq 1,\ a_{1,3} = x_3,\ b_{1,3} = 1,$$
$$(\forall\ k:\ 4 \leq k \leq m)\ a_{1,k} = b_{1,k} = 0.$$

The below constraint sets simulate the Speedup and Slowdown Rules:

Speedup Rule Constraints. For the ith line where $i > 1$ and $A[i] = 1$, we have

$$a_{i,1} \geq 1,\ a_{i,1} \geq a_{i-1,1} - x_i,\ b_{i,1} = b_{i-1,1},\ a_{i,2} = 0,\ b_{i,2} \geq x_i,\ b_{i,2} \geq b_{i-1,1},\ a_{i,3} \geq a_{i-1,2},$$
$$a_{i,3} \geq x_i,\ b_{i,3} \geq b_{i-1,2},\ (\forall\ k:\ 4 \leq k \leq m)\ a_{i,k} = a_{i-1,k-1}, b_{i,k} = b_{i-1,k-1}.$$

The constraints express that $\cdots\ {}^{b_2}(Q_2\ n^{a_2})^{b_1}\mathsf{DTS}[n^{a_1}]$ in the $(i - 1)$th line is replaced by

$$\cdots\ {}^{b_2}(Q_2\ n^{\max\{a_2,x\}})^{\max\{x,b_1\}}(Q_1\ n^0)^{b_1}\mathsf{DTS}[n^{\max\{a_1-x,1\}}]$$

in the ith line, where Q_1 is opposite to Q_2.

Slowdown Rule Constraints. For the ith line where $A[i] = 0$, the constraints are

$$a_{i,1} \geq c \cdot a_{i-1,1},\ a_{i,1} \geq c \cdot a_{i-1,2},\ a_{i,1} \geq c \cdot b_{i-1,1},\ a_{i,1} \geq c \cdot b_{i-1,2},\ b_{i,1} = b_{i-1,2}$$
$$(\forall\ k:\ 2 \leq k \leq m - 1)\ a_{i,k} = a_{i-1,k+1},\ b_{i,k} = b_{i-1,k+1},\ a_{i,m} = b_{i,m} = 0.$$

These express the replacement of $\cdots\ {}^{b_2}(Q_1 n^{a_2})^{b_1}\mathsf{DTS}[n^{a_1}]$ in the $(i - 1)$th line with

$$\cdots\ {}^{b_2}\mathsf{DTS}[n^{c\cdot\max\{a_1,a_2,b_1,b_2\}}]$$

in the ith line.

This concludes the description of the linear program. To find the largest c that still yields a feasible LP, we can simply binary search for it. The following summarizes this section.

Theorem 3.6. *Given an annotation of n lines, the best possible alternation-trading proof following the annotation can be determined up to n digits of precision, in* $\mathrm{poly}(n)$ *time.*

3.2. Results

Following the above formulation, we wrote proof search routines in Maple. Many millions of proof annotations were tried, including all those corresponding to prior work, with no success beyond the $2\cos(\pi/7)$ exponent. The best lower bounds followed a highly regular pattern; see the full version for more on this. We are led to:

Conjecture 3.7. *There is no alternation-trading proof that* $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTS}[n^c]$ *for all* $c > 2\cos(\pi/7)$.

Proving the conjecture seems currently out of reach. However, we can show:

Theorem 3.8. *There is no alternation-trading proof that* $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTS}[n^2]$.

A proof is in the full version. At a high level, the proof argues that any minimum length proof of a quadratic lower bound could be shortened, giving a contradiction.

Despite this bad news, the theorem prover did provide enough insight to aid in a new lower bound of $n^{2\cos(\pi/7)-o(1)}$ on the time-space product of any SAT algorithm.

Theorem 3.9. *Let $t(n)$ and $s(n)$ be bounded above by polynomials. Any algorithm solving SAT in time t and space s requires $t \cdot s = \Omega(n^{2\cos(\pi/7)-\varepsilon})$ for all $\varepsilon > 0$.*

These lower bounds have also been generalized to the QBF problem:

Theorem 3.10. *For all $k \geq 1$,* QBF_k *requires $\Omega(n^c)$ time on $n^{o(1)}$ space RAMs, where $c^3/k - c^2 - 2c + k < 0$.*

4. Discussion

We introduced a methodology for reasoning about alternation-trading proofs of lower bounds. It provides a generic means for computers to help us attack lower bound problems, and lets us establish limitations on known techniques. We now have a better understanding of what these techniques can and cannot do, and a tool for addressing future problems. Previously, the problem of setting parameters to achieve a good lower bound was a highly technical exercise. Our work should facilitate further research: once a new speedup or slowdown lemma is found, one only needs to find the relevant linear programming formulation to begin understanding its power. We conclude with two open-ended problems.

(1) *Establish tight limitations for alternation-trading proofs.* That is, show that the best possible alternation-trading proofs match those we have provided. Our computer search results have been met with healthy skepticism. It is critical to verify these perceived limitations with formal proof. We have managed to prove non-trivial limitations; it is possible that the ideas in those can be extended.

(2) *Discover new ingredients to add to the framework.* One possibility is to find new separation results that lead to new contradictions. Another is to find improved Speedup and/or Slowdown Lemmas. The Slowdown Lemmas are the “blandest” of the ingredients, in that they are the most elementary (and they relativize).

Acknowledgements. I am grateful to my thesis committee for their invaluable feedback on my PhD thesis, which included preliminary results on this work. Thanks to Scott Aaronson for useful discussions about irrelativization, and thanks to the STACS referees for very thoughtful comments.

References

[AKRRV01] E. Allender, M. Koucky, D. Ronneburger, S. Roy, and V. Vinay. Time-space tradeoffs in the counting hierarchy. In *Proc. IEEE Conference on Computational Complexity (CCC)*, 295–302, 2001.

[CKS81] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *JACM* 28(1):114–133, 1981.

[Coo88] S. A. Cook. Short propositional formulas represent nondeterministic computations. *IPL* 26(5): 269-270, 1988.

[Der72] C. Derman. *Finite state Markov decision processes.* Academic Press, 1972.

[DvM06] S. Diehl and D. van Melkebeek. Time-space lower bounds for the polynomial-time hierarchy on randomized machines. *SIAM J. Computing* 36: 563-594, 2006.

[DvMW09] S. Diehl, D. van Melkebeek, and R. Williams. An improved time-space lower bound for tautologies. In *Proc. of Computing and Combinatorics (COCOON)*, Springer LNCS 5609, 429–438, 2009.

[For97] L. Fortnow. Nondeterministic polynomial time versus nondeterministic logarithmic space. In *Proc. IEEE Conference on Computational Complexity (CCC)*, 52–60, 1997.

[FvM00] L. Fortnow and D. van Melkebeek. Time-Space Tradeoffs for Nondeterministic Computation. In *Proc. IEEE Conference on Computational Complexity (CCC)*, 2–13, 2000.

[FLvMV05] L. Fortnow, R. Lipton, D. van Melkebeek, and A. Viglas. Time-Space Lower Bounds for Satisfiability. *JACM* 52(6):835–865, 2005.

[HLMW86] J. Y. Halpern, M. C. Loui, A. R. Meyer, and D. Weise. On Time versus Space III. *Mathematical Systems Theory* 19(1):13–28, 1986.

[HPV77] J. Hopcroft, W. Paul, and L. Valiant. On time versus space. *JACM* 24(2):332–337, 1977.

[Kan83] R. Kannan. Alternation and the power of nondeterminism. In *Proc. ACM STOC*, 344–346, 1983.

[Kan84] R. Kannan. Towards separating nondeterminism from determinism. *Mathematical Systems Theory* 17(1):29–45, 1984.

[LV99] R. J. Lipton and A. Viglas. On the complexity of SAT. In *Proc. IEEE FOCS*, 459–464, 1999.

[Lou80] M. C. Loui. Simulations among multidimensional Turing machines. Ph.D. Thesis, Massachusetts Institute of Technology TR-242, 1980.

[MS87] W. Maass and A. Schorr. Speed-up of Turing machines with one work tape and a two-way input tape. *SIAM J. Computing* 16(1):195–202, 1987.

[vM04] D. van Melkebeek. Time-space lower bounds for NP-complete problems. In *Current Trends in Theoretical Computer Science* 265–291, World Scientific, 2004.

[vM07] D. van Melkebeek. A survey of lower bounds for satisfiability and related problems. *Foundations and Trends in Theoretical Computer Science* 2(3):197–303, 2007.

[vMR05] D. van Melkebeek and R. Raz. A time lower bound for satisfiability. *TCS* 348(2-3):311–320, 2005.

[vMW07] D. van Melkebeek and T. Watson. A quantum time-space lower bound for the counting hierarchy. Technical Report 1600, Department of Computer Sciences, University of Wisconsin-Madison, 2007.

[Nep70] V. Nepomnjascii. Rudimentary predicates and Turing calculations. *Soviet Math. Doklady* 11:1462–1465, 1970.

[PR81] W. Paul and R. Reischuk. On time versus space II. *JCSS* 22:312–327, 1981.

[PPST83] W. Paul, N. Pippenger, E. Szemeredi, and W. Trotter. On determinism versus nondeterminism and related problems. In *Proc. IEEE FOCS*, 429–438, 1983.

[Sch78] C. Schnorr. Satisfiability is quasilinear complete in NQL. *JACM* 25(1):136–145, 1978.

[Tou01] I. Tourlakis. Time-space tradeoffs for SAT on nonuniform machines. *JCSS* 63(2):268–287, 2001.

[Vio09] E. Viola. On approximate majority and probabilistic time. *Computational Complexity* 18(3):337–375, 2009.

[Wil06] R. Williams. Inductive time-space lower bounds for SAT and related problems. *Computational Complexity* 15:433–470, 2006.

[Wil07] R. Williams. Algorithms and resource requirements for fundamental problems. Ph.D. Thesis, Carnegie Mellon University, CMU-CS-07-147, August 2007.

[Wil08] R. Williams. Time-space tradeoffs for counting NP solutions modulo integers. *Computational Complexity* 17(2):179–219, 2008.