ARCoSS

LNCS 7464

Branislav Rovan
Vladimiro Sassone
Peter Widmayer (Eds.)

Mathematical Foundations of Computer Science 2012

37th International Symposium, MFCS 2012
Bratislava, Slovakia, August 2012
Proceedings

Springer

Lecture Notes in Computer Science 7464

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Advanced Research in Computing and Software Science

Subline of Lectures Notes in Computer Science

Branislav Rovan Vladimiro Sassone
Peter Widmayer (Eds.)

Mathematical Foundations of Computer Science 2012

37th International Symposium, MFCS 2012
Bratislava, Slovakia, August 27-31, 2012
Proceedings

Springer

Volume Editors

Branislav Rovan
Comenius University
Department of Computer Science
Mlynska Dolina, 84248 Bratislava, Slovakia
E-mail: rovan@dcs.fmph.uniba.sk

Vladimiro Sassone
University of Southampton
Department of Electronics and Computer Science
Southampton SO17 1BJ, UK
E-mail: vs@ecs.soton.ac.uk

Peter Widmayer
ETH Zürich
Institute of Theoretical Computer Science
CAB H 39.2, Universitätstrasse 6, 8092 Zürich, Switzerland
E-mail: widmayer@inf.ethz.ch

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-32588-5 e-ISBN 978-3-642-32589-2
DOI 10.1007/978-3-642-32589-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F.2, G.2, G.1, F.1, F.4.1, E.1, D.2.4, F.4.3, I.2.3-4, G.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The 37th International Symposium on Mathematical Foundations of Computer Science, MFCS 2009, was held in Bratislava (Slovakia) during August 27–31, 2012. It took place 40 years after the first MFCS that was held in 1972 in Jabłonna, Poland. This volume contains eight invited and 63 contributed papers presented at the symposium. The contributed papers were selected by the Program Committee out of a total of 162 submissions.

MFCS 2012 was organized by the Slovak Society for Computer Science and the Faculty of Mathematics, Physics and Informatics of the Comenius University in Bratislava. It was supported by the European Association for Theoretical Computer Science. We acknowledge with gratitude the support of all these institutions.

The series of MFCS symposia has a well-established tradition dating back to 1972. The aim is to encourage high-quality research in all branches of theoretical computer science, and to bring together researchers who do not usually meet at specialized conferences. The symposium is organized on a rotating basis in Poland, Czech Republic, and Slovakia. The previous meetings took place in Jabłonna 1972, Štrbské Pleso 1973, Jadwisin 1974, Mariánske Lázně 1975, Gdańsk 1976, Tatranská Lomnica 1977, Zakopane 1978, Olomouc 1979, Rydzyna 1980, Štrbské Pleso 1981, Prague 1984, Bratislava 1986. Karlovy Vary 1988, Porąbka-Kozubnik 1989, Banská Bystrica 1990, Kazimierz Dolny 1991, Prague 1992, Gdańsk 1993, Košice 1994, Prague 1995, Kraków 1996, Bratislava 1997, Brno 1998, Szklarska Poręba 1999, Bratislava 2000, Mariánske Lázně 2001, Warsaw 2002, Bratislava 2003, Prague 2004, Gdańsk 2005, Stará Lesná 2006, Český Krumlov 2007, Toruń 2008, Nový Smokovec 2009, Brno 2010, Warsaw 2011.

The 2012 meeting added a new page to this history, which was possible owing to the effort of many people.

We would like to thank the invited speakers Georg Gottlob (University of Oxford), Rolf Niedermeier (TU Berlin), Antonino Salibra (University of Venice), Nicole Schweikardt (Goethe University of Frankfurt), Esko Ukkonen (University of Helsinki), Igor Walukiewicz (University of Bordeaux), Gerhard J. Woeginger (TU Eindhoven), and Mihalis Yannakakis (Columbia University) for presenting their work to the audience of MFCS 2012. The papers provided by the invited speakers appear at the beginning of this volume. We thank all authors who have submitted their papers for consideration. Many thanks go to the Program Committee, and to all external referees, for their hard work in evaluating the papers. The work of the Program Committee was carried out using the EasyChair system, and we gratefully acknowledge this contribution.

Special thanks are due to the Organizing Committee led by Vanda Hambálková and Dana Pardubská.

June 2012

Branislav Rovan
Vladimiro Sassone
Peter Widmayer

Conference Organization

Conference Chair

Branislav Rovan

Program Chairs

Vladimiro Sassone
Peter Widmayer

Program Committee

Gilles Barthe	IMDEA, Madrid, Spain
Lars Birkedal	ITU, Copenhagen, Denmark
Michele Bugliesi	University of Ca' Foscari, Venice, Italy
Giuseppe Castagna	CNRS Paris, France
Jeremie Chalopin	University of Marseille, France
Krishnendu Chatterjee	IST Austria
Corina Cirstea	University of Southampton, UK
Andrea Clementi	University of Rome, Tor Vergata, Italy
Artur Czumaj	University of Warwick, UK
Alfredo De Santis	University of Salerno, Italy
Josep Diaz	UPC Barcelona, Spain
Thomas Erlebach	University of Leicester, UK
Michael Fellows	CDU, Darwin, Australia
Maribel Fernandez	King's College, London, UK
Marcelo Fiore	University of Cambridge, UK
Rudolf Fleischer	German University of Technology, Oman
Fedor Fomin	University of Bergen, Norway
Yuxi Fu	SJTU, Shanghai, China
Stefan Funke	University of Stuttgart, Germany
Leszek Gasieniec	University of Liverpool, UK
Subir Ghosh	TIFR, Mumbai, India
James Harland	RMIT, Melbourne, Australia
Lane Hemaspaandra	University of Rochester, USA
Giuseppe F. Italiano	University of Rome, Tor Vergata, Italy
Kazuo Iwama	Kyoto University, Japan
Juhani Karhumaki	University of Turku, Finland
Stefan Katzenbeisser	TU Darmstadt, Germany
Adrian Kosowski	INRIA Bordeaux, France
Jan Kratochvil	Charles University of Prague, Czech Republic

Jan van Leeuwen	University of Utrecht, The Netherlands
Leonid Libkin	University of Edinburgh, UK
Pasquale Malacaria	Queen Mary University, London, UK
Bernard Mans	Macquarie University, Sydney, Australia
Mohamed Reza Mousavi	TU Eindhoven, The Netherlands
Petra Mutzel	TU Dortmund, Germany
Mogens Nielsen	Aarhus University, Denmark
Leszek Pacholski	University of Wroclaw, Poland
Gennaro Parlato	University of Southampton, UK
Paolo Penna	University of Salerno, Italy
Giovanni Pighizzini	University of Milan, Italy
Grigore Rosu	University of Illinois, Urbana, USA
Branislav Rovan (Conference Chair)	Comenius University of Bratislava, Slovakia
Piotr Sankowski	Warsaw University, Poland
Vladimiro Sassone (Chair)	University of Southampton, UK
Yufei Tao	Chinese University, Hong Kong
Lidia Tendera	University of Opole, Poland
P.S. Thiagarajan	National University, Singapore
Gyorgy Turan	University of of Illinois, Chicago, USA
Peter Widmayer (Chair)	ETH Zurich, Switzerland
Masafumi Yamashita	Kyushu University, Fukuoka, Japan

Local Organization

Vanda Hambálková
Jaroslav Janáček
Dana Pardubská

Additional Reviewers

Aaronson, Scott
Aceto, Luca
Agrawal, Manindra
Akshay, S.
Al-Homaimeedi, Abiar
Albarelli, Andrea
Alvarez, Carme
Asavoae, Irina Mariuca
Asavoae, Mihail
Auletta, Vincenzo
Bacławski, Krystian
Baskaran, Muthu
Berger, Martin
Bernasconi, Anna
Berwanger, Dietmar
Bevern, René Van
Bilò, Davide
Biondi, Fabrizio
Boreale, Michele
Bournez, Olivier
Briquel, Irenee
Broadbent, Christopher
Brock-Nannestad, Taus
Bucci, Michelangelo
Calamoneri, Tiziana
Calzavara, Stefano
Capobianco, Silvio
Carpi, Arturo
Chadha, Rohit
Charatonik, Witold
Chen, Yijia
Chrobak, Marek
Cibulka, Josef
Cicalese, Ferdinando
Ciobaca, Stefan
Cirstea, Horatiu
Cordeiro, Lucas
Cori, Robert
Cygan, Marek
Czubak, Adam
D'Osualdo, Emanuele
Damaschke, Peter

Delbianco, Germán
Dell, Holger
Delzanno, Giorgio
Deng, Yuxin
Di Ianni, Miriam
Diakonikolas, Ilias
Doyen, Laurent
Drange, Pål Grønås
Dziubiński, Marcin
Eades, Peter
Ehrhard, Thomas
Ehsani, Shayan
Eisner, Jochen
Elbassioni, Khaled
Emmi, Michael
Enea, Constantin
Escada, Ana
Faella, Marco
Faliszewski, Piotr
Ferraioli, Diodato
Figueira, Diego
Fontaine, Gaelle
Formenti, Enrico
Fusco, Emanuele Guido
Gal, Anna
Ganty, Pierre
Gawrychowski, Pawel
Genest, Blaise
Giotis, Ioannis
Glasner, Yair
Goldwurm, Massimiliano
Golovach, Petr
Gonçalves, Daniel
Grabner, Peter
Graça, Daniel
Grochow, Joshua
Guo, Jiong
Gurevich, Yuri
Gutwenger, Carsten
Habermehl, Peter
Hague, Matthew
Halava, Vesa
Hansen, Kristoffer Arnsfelt
Harju, Tero
Heule, Marijn
Heußner, Alexander
Hildebrandt, Thomas
Hivert, Florent
Hoogeboom, Hendrik Jan
Horiyama, Takashi
Horvath, Tamas
Ilie, Lucian
Ivan, Szabolcs
Jain, Sanjay
Jansson, Jesper
Jez, Artur
Jiang, Ying
Jukna, Stasys
Kamiyama, Naoyuki
Kari, Jarkko
Kijima, Shuji
Kinder, Johannes
Konecny, Michal
Kopylova, Evguenia
Koubek, Vaclav
Krenn, Stephan
Kreutzer, Stephan
Kriege, Nils
Kubicki, Grzegorz
Kulkarni, Raghav
Kunc, Michal
Kurka, Petr
Kutsia, Temur
La Torre, Salvatore
Lavaee, Rahman
Lin, Anthony Widjaja
Liskiewicz, Maciej
Loreti, Michele
Luttik, Bas
Löding, Christof
M.S., Ramanujan
Maffezioli, Paolo
Maneth, Sebastian
Markou, Euripides
Martens, Wim
Martin, Russell
Mathieson, Luke
Mercas, Robert
Mereghetti, Carlo
Michaliszyn, Jakub
Mihalak, Matus
Milosavljevic, Nikola
Miltersen, Peter Bro
Mitsche, Dieter
Mogelberg, Rasmus
Monaco, Gianpiero
Montanaro, Ashley
Moore, Brandon Michael
Mulzer, Wolfgang
Murano, Aniello
Murawski, Andrzej
Naor, Seffi
Navarra, Alfredo
Nisse, Nicolas
Norman, Gethin
Ochem, Pascal
Okhotin, Alexander
Oliva, Paulo
Olmedo, Federico
Otop, Jan
Pal, Sudebkumar
Palano, Beatrice
Pasquale, Francesco
Paulusma, Daniël
Peng, Jing
Perelli, Giuseppe
Petri, Gustavo
Pieris, Andreas
Pilipczuk, Marcin
Pilipczuk, Michal
Pinaud, Bruno
Pirillo, Giuseppe
Polak, Libor
Potapov, Igor
Pouly, Amaury
Pribavkina, Elena
Provillard, Julien
Puzynina, Svetlana
Péchoux, Romain
R., Arjun
Rahman, Md. Saidur
Raman, Rajeev
Rampersad, Narad

Rao, Michael
Ravelomanana, Vlady
Rebagliati, Nicola
Reutter, Juan
Reyzin, Lev
Richard, Gaetan
Rota Bulò, Samuel
Rubin, Sasha
Saarela, Aleksi
Saivasan, Prakash
Salo, Ville
Sangnier, Arnaud
Sau, Ignasi
Scarpa, Giannicola
Schwoon, Stefan
Sciortino, Marinella
Seki, Shinnosuke
Semaev, Igor
Serna, Maria
Seto, Kazuhisa
Shpilka, Amir
Siddiqui, Junaid Haroon
Silvestri, Riccardo
Stefanescu, Andrei
Stephan, Frank
Storandt, Sabine
Suchy, Ondra
Swamy, Chaitanya
Szorenyi, Balazs
Talbot, Jean-Marc
Tamaki, Suguru
Tani, Seiichiro
Thapper, Johan
Theyssier, Guillaume
Thraves, Christopher
Turon, Aaron
Tzevelekos, Nikos
Uno, Yushi
Uznanski, Przemyslaw
van Leeuwen, Erik Jan
Varacca, Daniele
Ventre, Carmine
Volkov, Mikhail
Vollmer, Heribert
Vrgoc, Domagoj
Wasowski, Andrzej
Watson, Thomas
Weil, Pascal
Weiss, Armin
Wieczorek, Piotr
Wilkinson, Toby
Woeginger, Gerhard J.
Wozny, Pawel
Wrona, Michał
Yamauchi, Yukiko
Yao, Penghui
Zadimoghaddam, Morteza
Zamboni, Luca
Zey, Bernd
Zhang, Guochuan
Zhou, Xiaocong
Zimand, Marius

Table of Contents

On the Complexity of Ontological Reasoning under Disjunctive Existential Rules

Georg Gottlob[1,2,3], Marco Manna[1,4], Michael Morak[1], and Andreas Pieris[1]

[1] Department of Computer Science, University of Oxford, UK
[2] Oxford-Man Institute of Quantitative Finance, University of Oxford, UK
[3] Institute for the Future of Computing, Oxford Martin School, UK
[4] Department of Mathematics, University of Calabria, Italy
{georg.gottlob,michael.morak,andreas.pieris}@cs.ox.ac.uk,
manna@mat.unical.it

Abstract. Ontology-based data access is an emerging yet powerful technology that allows to enhance a classical relational database with an ontology in order to infer new intensional knowledge. Recently, Datalog+/- was introduced with the purpose of providing tractable reasoning algorithms for expressive ontology languages. In this framework, Datalog is extended by features such as existential quantification in rule heads, and at the same time the rule syntax is restricted to guarantee decidability, and also tractability, of relevant reasoning tasks. In this paper, we enrich Datalog even more by allowing not only existential quantification but also disjunction in rule heads, and we investigate the complexity of reasoning under the obtained formalism.

1 Introduction

Ontological reasoning is a fundamental task in the Semantic Web, where the information present in the web is annotated in order to be machine-readable. Semantic Web ontologies are modeled using logical formalisms, such as W3Cs standard OWL 2 DL ontology language[1] which is built on *Description Logics (DLs)* [6]. DLs are decidable fragments of first-order logic based on concepts (classes of objects) and roles (binary relations between concepts). A large corpus of works in DLs has focused on the problems of consistency (whether a knowledge base is consistent or satisfiable), instance checking (whether a certain object is an instance of a concept), and logical entailment (whether a certain constraint is logically implied by an ontology). The last few years, the attention has shifted on the problem of query answering. A notable example is the *DL-Lite* family of DLs [18,30], which forms the OWL 2 QL profile[2] of OWL 2 DL, that offers flexible, natural and expressive languages, and at the same time keeps query answering highly tractable and scalable to large data sets.

Recently, the DL-Lite family (together with other well-known DLs such as the *DLR-Lite* family [17] and $\mathcal{EL}$ [4]) has been embedded into an expressive framework called *Datalog*$^{\pm}$ [14]. The Datalog$^{\pm}$ family has been proposed with the purpose of providing

[1] http://www.w3.org/TR/owl2-overview/

[2] http://www.w3.org/TR/owl2-profiles/

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 1–18, 2012.

tractable reasoning algorithms for more general ontology languages. These languages are based on Datalog rules[3] that allow for existentially quantified variables in rule heads, in the same fashion as Datalog with *value invention* [11]. Such rules are known as *tuple-generating dependencies (TGDs)* in the database literature [9]. In particular, TGDs are implications between conjunctions of atoms. They essentially say that some tuples in an instance I imply the presence of some other tuples in I. For example, the TGD

$$\forall E \forall D\, emp(E) \wedge mgr(E, D) \rightarrow \exists E'\, emp(E') \wedge works(E', D) \wedge reports(E', E)$$

expresses the following: "if an employee E is the manager of a department D, then there is an employee E' who works in D and reports to E." The absence of value invention, thoroughly discussed in [29], is the main shortcoming of Datalog in modeling ontologies, and even conceptual data formalisms such as UML class diagrams [10] and ER schemata [19]. The addition of value invention in the form of existential quantification in rule heads constitutes a crucial step towards the bridging of the gap between ontology languages and Datalog, and opens new horizons in ontological reasoning.

Unfortunately, the addition of existential quantification in Datalog easily leads to undecidability of the most basic reasoning tasks [9,12]. Currently, an important research direction in the general area of knowledge representation and reasoning is to identify expressive Datalog$^{\pm}$ formalisms (or, equivalently, classes of TGDs) for which the basic reasoning services are decidable. Moreover, to be able to work with very large data sets, it is desirable not only that reasoning is decidable, but also tractable in data complexity (i.e., in the size of the database). The main syntactic paradigms which guarantee the above desirable properties are *weak-acyclicity* [22], *guardedness* [12], *stickiness* [15] and *shyness* [27]. Several attempts have been conducted towards the identification of even more expressive formalisms, by extending or combining the above classes (see, e.g., [7,16,26,28]).

The aforementioned Datalog-based formalisms, in contrast to expressive DL-based ontology languages, are not powerful enough for nondeterministic reasoning. For instance, in the well-known description logic $\mathcal{ELU}$, that is, $\mathcal{EL}$ extended with disjunction (see, e.g., [5]), it is possible to state simple and natural statements such as "the parent of a father is a grandfather *or* a grandmother" using the following axiom:

$$\exists parentOf.father \sqsubseteq \exists grandfatherOf.\top \sqcup \exists grandmotherOf.\top$$

This axiom can be expressed by a guarded TGD extended with disjunction as follows:

$$\forall X \forall Y\, parentOf(X, Y) \wedge father(Y) \rightarrow \exists Z\, gfatherOf(X, Z) \vee gmotherOf(X, Z)$$

Obviously, to represent such kind of knowledge, we need to extend the existing classes of TGDs by allowing disjunctive statements in the head. *Disjunctive Datalog*, that is, the variant of Datalog where disjunction may appear in rule heads, has been thoroughly investigated by Eiter et al. [21]. The data complexity of reasoning under guarded-based classes of TGDs extended with disjunction has been recently studied in [2].

Research Challenge. It is the precise aim of the current work to better understand the problem of reasoning under *disjunctive TGDs (DTGDs)*, that is, TGDs that allow for

[3] A Datalog program is a set of universally quantified function-free Horn clauses (see, e.g., [1]).

disjunction in their heads, and to investigate the complexity of reasoning under existing classes of TGDs extended with disjunction. In particular, we concentrate on the problem of *atom entailment* defined as follows: given an extensional database D, an ontology Σ constituted by DTGDs, and a relational atom $\underline{a}$, decide whether each model of D and Σ, i.e., each instance that contains D and satisfies Σ, entails $\underline{a}$, denoted $D \cup \Sigma \models \underline{a}$. Notice that the problem of instance checking is a special case of atom entailment.

Ontological reasoning adheres to the standard logical semantics of entailment ($\models$), which denotes entailment under arbitrary, not necessarily finite, models. This implies, in general, that a database and a set of DTGDs admit infinitely many models, where each of them can be of infinite size; in fact, this holds already for TGDs due to the existential quantification. Interestingly, in the case of TGDs, it is always possible to construct the so-called *universal model*, which can be seen as a representative of all the other models, by applying a well-known procedure called *chase* (see, e.g., [20,24]). Roughly, the chase adds new atoms to the given database as dictated by the given TGDs, possibly involving labeled null values as witnesses for the existentially quantified variables, until the final result satisfies all the TGDs. Hence, for reasoning purposes, instead of considering all the models of an ontological theory constituted by TGDs, we can concentrate on the universal model.

The situation changes dramatically if we consider DTGDs. In fact, there is no single model that acts as a representative of all the other models. The notion of universal model has been recently extended to the disjunctive case by introducing *universal model sets* [2]. It was shown that, given an extensional database D and a set Σ of DTGDs, by applying a procedure similar to the chase, one can build a set $P_{D,\Sigma}$ of ground rules, called the *instantiation* of D and Σ. The models of $P_{D,\Sigma}$ constitute the universal model set for D and Σ. Nevertheless, $P_{D,\Sigma}$ admits, in general, infinitely many models, that have to be considered for ontological reasoning purposes. From the above discussion, it is clear that atom entailment (and other reasoning tasks) is becoming even more complicated and challenging if we consider DTGDs instead of TGDs. Obviously, correct and terminating algorithms require methods for reasoning over infinite structures without explicitly building them.

Summary of Contributions. We investigate the complexity of atom entailment under guarded and linear DTGDs. Recall that guarded TGDs is one of the main classes of TGDs that guarantees decidability, and also tractability in data complexity, of reasoning [12]. A TGD is *guarded* if it has an atom in its body that contains all the

Table 1. Complexity of atom entailment under guarded and linear DTGDs

Formalism	Data Complexity	DTGDs of fixed size	Combined Complexity
Guarded DTGDs	**coNP**-complete UB: [8] LB: [2]	**EXPTIME**-complete UB: [23] LB: [12]	**2EXPTIME**-complete UB: [23] LB: [12]
Linear DTGDs	in $\mathbf{AC}_0$ UB: Thm. 3	**PTIME**-complete UB: Thm. 2 LB: Thm. 5	**EXPTIME**-complete UB: Thm. 2 LB: Thm. 4

body-variables, while a TGD is *linear* if it has only one body atom (and thus it is guarded). The complexity results established in this paper are presented in Table 1; we have indicated, in each cell, where to find the corresponding results (UB and LB stand for upper bound and lower bound, respectively). A summary of our contribution follows:

– We show that atom entailment under guarded DTGDs is **coNP**-complete in data complexity, **EXPTIME**-complete in case of DTGDs of fixed size, and **2EXPTIME**-complete in combined complexity, i.e., the complexity calculated by considering, together with the database, also the set of DTGDs as part of the input. The upper bounds are obtained from existing results on satisfiability of logical theories that fall in the guarded fragment of first-order logic [8,23]. In particular, we show that atom entailment under guarded DTGDs can be reduced to the problem of unsatisfiability of guarded first-order theories. The lower bounds are inherited immediately from existing results on atom entailment under guarded DTGDs. We also show that every $\mathcal{ELU}$ ontology [5] can be rewritten into an equivalent set (w.r.t. atom entailment) of guarded DTGDs.
– We show that atom entailment under linear DTGDs is in $\mathbf{AC}_0$ in data complexity[4] (improving the **LOGSPACE** upper bound established in [2]), **PTIME**-complete in the case of DTGDs of fixed size, and **EXPTIME**-complete in combined complexity. These are novel complexity results, and constitute the main contribution of this paper. The $\mathbf{AC}_0$ upper bound is obtained by establishing that atom entailment under linear DTGDs can be reduced to the evaluation of a first-order query over a database [32]. The remaining two upper bounds are obtained by giving an alternating algorithm which runs in logarithmic space if the DTGDs have fixed size, and in polynomial space in general. The lower bounds are established by simulating the behavior of an alternating logarithmic space (resp., polynomial space) Turing machine by means of linear DTGDs.

2 The Framework

In this section, we present background material necessary for this paper. We recall some basics on relational databases, we introduce disjunctive tuple-generating dependencies, and we discuss the problem tackled in this work, i.e., atom entailment.

Technical Definitions. We define the following pairwise disjoint (countably infinite) sets of symbols: a set Γ of *constants* (constituting the "normal" domain of a database), a set Γ_N of *labeled nulls* (used as placeholders for unknown values, and thus can be also seen as globally existentially quantified variables), and a set Γ_V of regular *variables* (used in dependencies). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. We assume a fixed well-ordering of $\Gamma \cup \Gamma_N$ such that every value in Γ_N follows all those in Γ. We denote by $\mathbf{X}$ sequences (or sets, with a slight abuse of notation) of variables $X_1, \ldots, X_k$, with $k \geqslant 0$. Throughout, let $[n] = \{1, \ldots, n\}$, for any integer $n \geqslant 1$.

A *relational schema* $\mathcal{R}$ (or simply *schema*) is a set of *relational symbols* (or *predicates*), each with its associated arity. We write r/n to denote that the predicate r has

[4] This is the complexity class of recognizing words in languages defined by constant-depth Boolean circuits with an (unlimited fan-in) AND and OR gates.

arity n. A *term* t is a constant, null, or variable. An *atomic formula* (or simply *atom*) has the form $r(t_1, \dots, t_n)$, where r/n is a relation, and $t_1, \dots, t_n$ are terms. For an atom $\underline{a}$, we denote as $terms(\underline{a})$ and $var(\underline{a})$ the set of its terms and the set of its variables, respectively. These notations naturally extend to sets and conjunctions of atoms. The arity of an atom is the arity of its predicate. An atom is called a *fact* if all of its terms are constants of Γ. Conjunctions and disjunctions of atoms are often identified with the sets of their atoms. An *instance* I for a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $r(\mathbf{t})$, where $r/n \in \mathcal{R}$ and $\mathbf{t} \in (\Gamma \cup \Gamma_N)^n$. A *database* D is a finite instance such that $terms(D) \subset \Gamma$.

A *substitution* from a set of symbols S to a set of symbols S' is a function $h : S \to S'$ defined as follows: $\varnothing$ is a substitution (empty substitution), and if h is a substitution, then $h \cup \{s \to s'\}$ is a substitution, where $s \in S$ and $s' \in S'$. If $s \to s' \in h$, then we write $h(s) = s'$. The *restriction* of h to $T \subseteq S$, denoted $h|_T$, is the substitution $h' = \{t \to h(t) \mid t \in T\}$. A *homomorphism* from a set of atoms A to a set of atoms A' is a substitution $h : \Gamma \cup \Gamma_N \cup \Gamma_V \to \Gamma \cup \Gamma_N \cup \Gamma_V$ such that: if $t \in \Gamma$, then $h(t) = t$, and if $r(t_1, \dots, t_n) \in A$, then $h(r(t_1, \dots, t_n)) = r(h(t_1), \dots, h(t_n)) \in A'$. An *isomorphism* between A and A' is a bijective homomorphism h such that $h(A) = A'$, h^{-1} is a homomorphism, and $h^{-1}(A') = A$.

Disjunctive TGDs. The set of (quantifier-free, positive) AND-OR formulas over a relational schema $\mathcal{R}$ is defined as the set of all (finite) formulas over $\mathcal{R}$ containing $\wedge$ and $\vee$. It is well-known that each AND-OR formula can be converted into an equivalent one in *disjunctive normal form (DNF)*. Given an AND-OR formula ψ, we denote by $\mathit{DNF}(\psi)$ the set of formulas in disjunctive normal form into which ψ can be converted. A *disjunctive tuple-generating dependency (DTGD)* σ over a schema $\mathcal{R}$ is a first-order formula $\forall \mathbf{X} \forall \mathbf{Y}\, \varphi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$, where φ is an AND formula over $\mathcal{R}$, ψ is an AND-OR formula over $\mathcal{R}$, and $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z} \subset \Gamma_V$. Formula φ is the *body* of σ, denoted as $\mathit{body}(\sigma)$, while ψ is the *head* of σ, denoted as $\mathit{head}(\sigma)$. We define the *size* of σ as the sum of the arities of its atoms. In the rest of the paper, for brevity, we will omit universal quantifiers in front of DTGDs, and use a comma for conjoining atoms instead of $\wedge$. Such σ is satisfied by an instance I, written as $I \models \sigma$, if the following holds: whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$, then there exists $\psi' \in \mathit{DNF}(\psi)$ and a homomorphism $h' \supseteq \{X \to t \mid X \to t \in h|_{\mathbf{X}},$ where $X \in \mathbf{X}$ and $t \in \Gamma \cup \Gamma_N\}$ such that h' maps at least one disjunct of ψ' into I. An instance I satisfies a set Σ of DTGDs, denoted $I \models \Sigma$, if $I \models \sigma$ for each $\sigma \in \Sigma$.

Atom Entailment under DTGDs. Given a database D for a schema $\mathcal{R}$, and a set Σ of DTGDs over $\mathcal{R}$, a *model* of D and Σ is an instance I for $\mathcal{R}$ such that $I \supseteq D$ and $I \models \Sigma$; let $\mathit{mods}(D, \Sigma)$ be the set of models of D and Σ. An atom $\underline{a}$ over $\mathcal{R}$ is *entailed* by D and Σ, denoted as $D \cup \Sigma \models \underline{a}$, if for each $M \in \mathit{mods}(D, \Sigma)$, there exists a homomorphism h_M such that $h_M(\underline{a}) \in M$. The central decision problem tackled in this work, called *atom entailment*, is defined as follows:

ATOM-ENTAILMENT

Instance : $\langle \underline{a}, D, \Sigma \rangle$, where $\underline{a}$ is an atom over a schema $\mathcal{R}$, D is a database for $\mathcal{R}$, and Σ is a set of DTGDs over $\mathcal{R}$.

Question : $D \cup \Sigma \models \underline{a}$?

It is well-known that the above problem is undecidable already for *tuple-generating dependencies (TGDs)*, i.e., DTGDs where the head is an AND formula. More precisely, atom entailment is undecidable even under a fixed set of TGDs [12]. The proof of this result hinges on the fact that, with appropriate input atoms, we can simulate the behavior of a deterministic Turing machine using a fixed set of TGDs. Also, as established recently [7], it is possible to encode any set of TGDs using a single TGD.

From the above discussion, we conclude that the recognition of expressive decidable classes of TGDs is a challenging problem. Nowadays, an important research objective is to identify more expressive formalisms under which atom entailment (and other reasoning tasks) is still decidable (see, e.g., [7,14,26]). Known decidable formalisms, which are of special interest for the current work, are the classes of guarded and linear TGDs [13]. A TGD σ is *guarded* if in $body(\sigma)$ there exists an atom $\underline{a}$ such that $var(\underline{a}) = var(body(\sigma))$, i.e., $\underline{a}$ contains all the universally quantified variables of σ. If $body(\sigma)$ contains a single atom, then σ is called *linear*; notice that a linear TGD is trivially guarded since the single body-atom contains all the universally quantified variables. Guarded and linear TGDs can be naturally extended to guarded and linear DTGDs, respectively. In the rest of the paper, we investigate the complexity of atom entailment under guarded and linear DTGDs. Following Vardi's taxonomy [31], the *data complexity* of atom entailment is the complexity calculated taking only the database as input, while the atom and the set of dependencies are considered fixed. The *combined complexity* is the complexity calculated considering as input, together with the database, also the atom and the set of dependencies.

Normalization of DTGDs. In order to simplify the technical definitions and proofs in the rest of the paper, a normal form of DTGDs is adopted. Given a set Σ of DTGDs, we denote by $\mathsf{Normalize}(\Sigma)$ the set of DTGDs obtained by transforming Σ as follows. First, we construct the set Σ' by applying exhaustively on Σ the following rules until each DTGD has in its head either a single atom or a disjunction of two atoms:

1. A DTGD of the form $\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi_1(\mathbf{X}_1,\mathbf{Z}_1) \wedge \psi_2(\mathbf{X}_2,\mathbf{Z}_2)$ is replaced by $\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r^\star(\mathbf{X},\mathbf{Z})$ and $r^\star(\mathbf{X},\mathbf{Z}) \rightarrow \psi_i(\mathbf{X}_i,\mathbf{Z}_i)$, for each $i \in [2]$, where $r^\star$ is an $|\mathbf{X} \cup \mathbf{Z}|$-ary auxiliary predicate not introduced so far.
2. A DTGD of the form $\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi_1(\mathbf{X}_1,\mathbf{Z}_1) \vee \psi_2(\mathbf{X}_2,\mathbf{Z}_2)$ is replaced by $\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \vee_{i\in[2]}\, r_i^\star(\mathbf{X}_i,\mathbf{Z}_i)$ and $r_i^\star(\mathbf{X}_i,\mathbf{Z}_i) \rightarrow \psi_i(\mathbf{X}_i,\mathbf{Z}_i)$, for each $i \in [2]$, where $r_i^\star$ is an $|\mathbf{X}_i \cup \mathbf{Z}_i|$-ary auxiliary predicate not introduced so far.

$\mathsf{Normalize}(\Sigma)$ is obtained by transforming Σ' in such a way that eventually each DTGD has at most one existentially quantified variable that occurs only once. In particular, for each $\sigma \in \Sigma'$, assuming that $var(body(\sigma)) \cap var(head(\sigma)) = \mathbf{X}$ and $var(head(\sigma)) \setminus var(body(\sigma)) = \{Z_1,\ldots,Z_n\}$, where $n \geqslant 1$, σ is replaced by

$$\begin{aligned} body(\sigma) &\rightarrow \exists Z_1\, r_1^\star(\mathbf{X},Z_1) \\ r_i^\star(\mathbf{X},Z_1,\ldots,Z_i) &\rightarrow \exists Z_{i+1}\, r_{i+1}^\star(\mathbf{X},Z_1,\ldots,Z_{i+1}), \quad \text{for each } i \in [n-1] \\ r_n^\star(\mathbf{X},Z_1,\ldots,Z_n) &\rightarrow head(\sigma), \end{aligned}$$

where $r_i^\star$ is an $(|\mathbf{X}|+i)$-ary auxiliary predicate, for each $i \in [n]$. It is easy to verify that, during the above construction, we introduce polynomially many auxiliary predicates, and the arity of those predicates is at most the maximum number of variables that appear in the head of a DTGD of Σ. Thus, $\mathsf{Normalize}(\Sigma)$ can be constructed in polynomial time. Finally, it is important to say that the normalized set is (w.r.t. atom entailment) equivalent to the original set. In the rest of the paper, we will state explicitly when we consider normalized sets of DTGDs.

3 Guarded Disjunctive Tuple-Generating Dependencies

The present section weaves together existing results, achieved in the areas of both logics and database theory, to establish the complexity of atom entailment under guarded DTGDs. For the upper bounds, we exploit the *guarded fragment of first-order logic* [3,23], while for the lower bounds we discuss relevant reductions that have already been provided for well-known subclasses of DTGDs.

Interestingly, every set Σ of guarded DTGDs can be translated in logarithmic space into an equivalent guarded first-order theory denoted as $\mathsf{GFO}(\Sigma)$. Given an instance $\langle \underline{a}, D, \Sigma \rangle$ of atom entailment, where Σ is a set guarded DTGDs, it is well-known that $D \cup \Sigma \models \underline{a}$ iff $\mathsf{GFO}(\Sigma) \wedge \psi_D \wedge \neg \underline{a}$ is unsatisfiable, where $\psi_D = \wedge_{\underline{d} \in D} \underline{d}$. Since the theory $\mathsf{GFO}(\Sigma) \wedge \psi_D \wedge \neg \underline{a}$ is guarded, we can exploit existing results regarding unsatisfiability of guarded formulas. The complexity of this problem was first investigated by Grädel [23]. In particular, it is in **2EXPTIME** in the general case and in **EXPTIME** in case of fixed arity. The former result was obtained by exhibiting an alternating exponential space algorithm. A deterministic version of this algorithm runs in time $\mathcal{O}(2^{(|\mathcal{R}|+|\psi|)\cdot\omega^{\omega}})$, where $\mathcal{R}$ is the set of predicates in the formula ψ, and ω is the maximum arity over all predicate symbols of $\mathcal{R}$. In the case of fixed arity, the same algorithm clearly runs in exponential time. In case of a fixed theory combined with an input database (data complexity) the same problem is in **NP** [8]. The idea behind this result can be summarized as follows. Consider a fixed guarded first-order formula ψ over a schema $\mathcal{R}$. The algorithm first pre-computes all possible maximal conjunctions of (pairwise non-isomorphic) literals, called *types*, over $\mathcal{R}$ that satisfy ψ. Since ψ is fixed, also the set of types is fixed, as well as the number of steps required to compute it. For example, if $\psi = \forall X(r(X) \rightarrow p(X))$, then its types are: $r(X) \wedge p(X)$, $\neg r(X) \wedge p(X)$, and $\neg r(X) \wedge \neg p(X)$. Next, given an input database ψ_D for $\mathcal{R}$, the algorithm guesses a suitable set B of constants appearing in ψ_D, constructs the set $S = \{\underline{c} \in \psi_D \mid \mathit{terms}(\underline{c}) \subseteq B\}$, and checks in polynomial time whether at least one type of ψ is consistent with S.

Let us now turn our attention to the lower bounds. In the general case, Calì et al. [12] proved that atom entailment is already **2EXPTIME**-hard for guarded TGDs, by simulating an alternating exponential space Turing machine. Moreover, in the same paper, it was also proved that if we consider predicates of fixed arity, the construction can be adapted to simulate an alternating linear space Turing machine. Regarding data complexity, Alviano et al. [2] recently showed that the well-known **coNP**-complete problem of deciding whether a 3-CNF formula is unsatisfiable can be reduced to atom entailment under the fixed set Σ of guarded DTGDs $\{\mathit{clause}(L_1, L_2, L_3, N_1, N_2, N_3) \rightarrow \vee_{i \in [3]} \mathit{assign}(L_i, N_i)\} \cup \{\mathit{assign}(L, N), \mathit{assign}(N, L) \rightarrow \mathit{invalid}(L)\}$. Given a propositional formula ψ in 3-CNF, for each clause of the form $l_1 \vee l_2 \vee l_3$ of ψ we add to the database D the fact $\mathit{clause}(l_1, l_2, l_3, \nu(l_1), \nu(l_2), \nu(l_3))$, where $\nu(l) = \neg x$ if $l = x$, and

$\nu(l) = x$ if $l = \neg x$. It can be shown that ψ is unsatisfiable iff $D \cup \Sigma \models \mathit{invalid}(L)$. The previous discussion is summarized in the following result.

Theorem 1. ATOM-ENTAILMENT *under guarded DTGDs is **2EXPTIME**-complete in combined complexity, **EXPTIME**-complete if the size of the DTGDs is fixed, and **coNP**-complete in data complexity.*

It is interesting to see that every set of $\mathcal{ELU}$ axioms [5] can be easily reduced to an equivalent set (w.r.t. atom entailment) of guarded DTGDs. A set of $\mathcal{ELU}$ axioms can be normalized in such a way that only assertions of the form given in Table 2 can appear (see, e.g., [4]). In the same table we give the translation τ of such axioms to guarded DTGDs. Using τ it is not difficult to show that instance checking under $\mathcal{ELU}$ knowledge bases can be reduced in logarithmic space to atom entailment under guarded DTGDs, while preserving the data complexity. It is well-known that instance checking under $\mathcal{ELU}$ ontologies is **coNP**-complete [25]. Guarded DTGDs are expressive enough to capture also the DL obtained by allowing in $\mathcal{ELU}$ role hierarchies and inverse roles.

4 Linear Disjunctive Tuple-Generating Dependencies

In this section, we study the complexity of atom entailment under linear DTGDs. We show that it is **EXPTIME**-complete in combined complexity, **PTIME**-complete if the DTGDs have fixed size, and in $\mathbf{AC}_0$ in data complexity. The first two upper bounds are obtained by giving an alternating algorithm which runs in polynomial space, and in logarithmic space if the DTGDs have fixed size. The $\mathbf{AC}_0$ upper bound is obtained by reducing atom entailment under linear DTGDs into first-order query evaluation. The lower bounds are established by simulating the behavior of an alternating polynomial space (resp., logarithmic space) Turing machine by means of linear DTGDs. It is important to say that the normalization algorithm defined in Section 2 preserves linearity.

4.1 Upper Bounds

Let us first concentrate on the upper bounds. We reduce an instance $\langle \underline{a}, D, \Sigma\rangle$ of atom entailment to the problem of checking the existence of a proof-tree of an atom $\underline{d} \in D$

Table 2. Translation of $\mathcal{ELU}$ to guarded DTGDs; A, B, C are concept names, R is a role name

$\mathcal{ELU}$ Axiom	Guarded DTGD
$A \sqsubseteq B$	$p_A(X) \rightarrow p_B(X)$
$A \sqcap B \sqsubseteq C$	$p_A(X), p_B(X) \rightarrow p_C(X)$
$A \sqsubseteq \exists R.B$	$p_A(X) \rightarrow \exists Y\, p_R(X,Y), p_B(Y)$
$\exists R.A \sqsubseteq B$	$p_R(X,Y), p_A(Y) \rightarrow p_B(X)$
$A \sqsubseteq B \sqcup C$	$p_A(X) \rightarrow p_B(X) \vee p_C(X)$

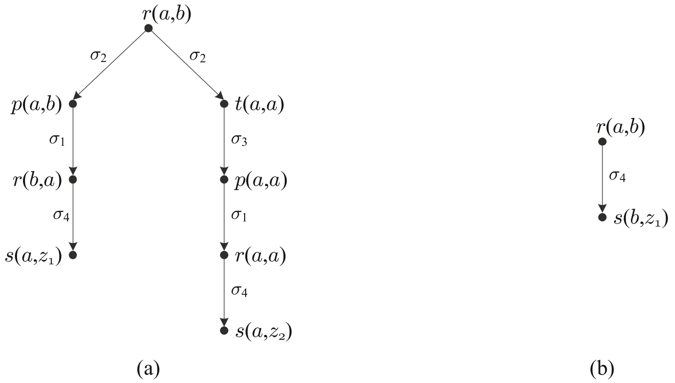


Fig. 1. Possible proof-trees of $r(a,b)$ and Σ

and Σ which is valid w.r.t. $\underline{a}$. Intuitively, such a tree encodes the part of each model of $D \cup \Sigma$ due to which $\underline{a}$ is entailed; the formal definition follows. Consider a fact $\underline{d}$, and a normalized set Σ of linear DTGDs. Let T be a binary tree $\langle N, E, \lambda_1, \lambda_2 \rangle$, where the nodes of N are labeled by λ_1 with atoms that can be formed using predicates occurring in Σ and terms of $terms(\underline{d}) \cup \Gamma_N$, and the edges of E are labeled by $\lambda_2 : E \rightarrow \Sigma$. We say that T is a *proof-tree* of $\underline{d}$ and Σ if:

- The root is labeled by $\underline{d}$.
- For each $v \in N$, there exists $\sigma \in \Sigma$ such that each edge of the form $(u, v) \in E$ is labeled by σ, and the out-degree of v is $|head(\sigma)|$.
- For each $v \in N$, if v has a single outgoing edge (v, u), which is labeled by $p(\mathbf{X}, \mathbf{Y}) \rightarrow \exists Z\, r(\mathbf{X}, Z)$, then there is a homomorphism h such that $h(p(\mathbf{X}, \mathbf{Y})) = \lambda_1(v)$, and there exists $h' = h|_{\mathbf{X}} \cup \{Z \rightarrow t \mid t \in \Gamma_N,\, t \notin terms(h(p(\mathbf{X}, \mathbf{Y})))\}$ such that $\lambda_1(u) = h'(r(\mathbf{X}, Z))$.
- For each $v \in N$, assuming that the outgoing edges of v are labeled by some σ without an existentially quantified variable, there is a homomorphism h such that $h(body(\sigma)) = \lambda_1(v)$, and $\{h(\underline{b}) \mid \underline{b} \in head(\sigma)\} = \{\lambda_1(u) \mid (v, u) \in E\}$.

A proof-tree T of $\underline{d}$ and Σ is *valid* w.r.t. to an atom $\underline{a}$ with $terms(\underline{a}) \subset \Gamma \cup \Gamma_V$ if, for each leaf u of T, there exists a homomorphism h such that u is labeled by $h(\underline{a})$.

Example 1. Consider the set Σ of linear DTGDs

$$\begin{array}{ll} \sigma_1 : p(X,Y) \rightarrow r(Y,X) & \sigma_2 : r(X,Y) \rightarrow p(X,Y) \vee t(X,X) \\ \sigma_3 : t(X,Y) \rightarrow p(X,Y) & \sigma_4 : r(X,Y) \rightarrow \exists Z\, s(Y,Z). \end{array}$$

Possible proof-trees of $r(a,b)$ and Σ are shown in Figure 1. In particular, tree (a) is valid w.r.t. $s(a,X)$ or $s(X,Y)$, while tree (b) is valid w.r.t. $s(b,X)$ or $s(X,Y)$. □

The following lemma, established in [2], shows that for atom entailment purposes under linear DTGDs, we are allowed to concentrate on a single database atom.

Lemma 1. *Given a database D, a normalized set Σ of linear DTGDs, and an atom $\underline{a}$ with $terms(\underline{a}) \subset \Gamma \cup \Gamma_V$, $D \cup \Sigma \models \underline{a}$ iff there exists $\underline{d} \in D$ such that $\{\underline{d}\} \cup \Sigma \models \underline{a}$.*

Although we can focus on a single database atom, in general we have to consider infinitely many models. The key idea underlying our approach is to use "representatives" of all these models in order to be able to encode them in a single structure, namely, the proof-tree defined above. This can be achieved by considering the skolemized version of the given set of DTGDs. Given a normalized set Σ of linear DTGDs, we define F as the set of *skolem functions* $\{f_\sigma \mid \sigma \in \Sigma\}$, where the arity of each f_σ is the number of universally quantified variables of σ. Let Σ_f denote the set of rules obtained from Σ by replacing each TGD σ of the form $p(\mathbf{X}, \mathbf{Y}) \rightarrow \exists Z\, r(\mathbf{X}, Z)$ with the rule $p(\mathbf{X}, \mathbf{Y}) \rightarrow r(\mathbf{X}, f_\sigma(\mathbf{X}, \mathbf{Y}))$. From well-known results on skolemization the following holds: given a database D, a normalized set Σ of linear DTGDs, and an atom $\underline{a}$ with $terms(\underline{a}) \subset \Gamma \cup \Gamma_V$, $D \cup \Sigma \models \underline{a}$ iff $D \cup \Sigma_f \models \underline{a}$. The set of *skolem terms* Γ_Σ is recursively defined as follows: each term of Γ belongs to Γ_Σ, and if $f_\sigma \in F$ has arity $n > 0$ and $t_1, \ldots, t_n$ are terms of Γ_Σ, then $f_\sigma(t_1, \ldots, t_n) \in \Gamma_\Sigma$. The notion of homomorphism naturally extends to atoms that contain functional terms of the form $f(t_1, \ldots, t_n)$, where each t_i is a variable of Γ_V or a skolem term; in particular, given a homomorphism h, $h(f(t_1, \ldots, t_n)) = f(h(t_1), \ldots, h(t_n))$.

Consider a normalized set Σ of linear DTGDs, an atom $\underline{c}$ with $terms(\underline{c}) \subset \Gamma_\Sigma$, and an instance M of $mods(\{\underline{c}\}, \Sigma_f)$. The *tree* of M is inductively defined as follows: $tree(M)$ is a rooted tree $\langle N, E, \lambda_1, \lambda_2 \rangle$, where N is the set of nodes, E is the edge set, $\lambda_1 : N \rightarrow M$ is a node-labeling function, and $\lambda_2 : E \rightarrow \Sigma_f$ is an edge-labeling function. The root of $tree(M)$ is labeled by $\underline{c}$. Consider a node $v \in N$, and a rule ρ of the form $\underline{b} \rightarrow \underline{b}_1 \vee \underline{b}_2$ or $\underline{b} \rightarrow \underline{b}_1$ such that there is a homomorphism h that maps $\underline{b}$ into $\lambda_1(v)$.

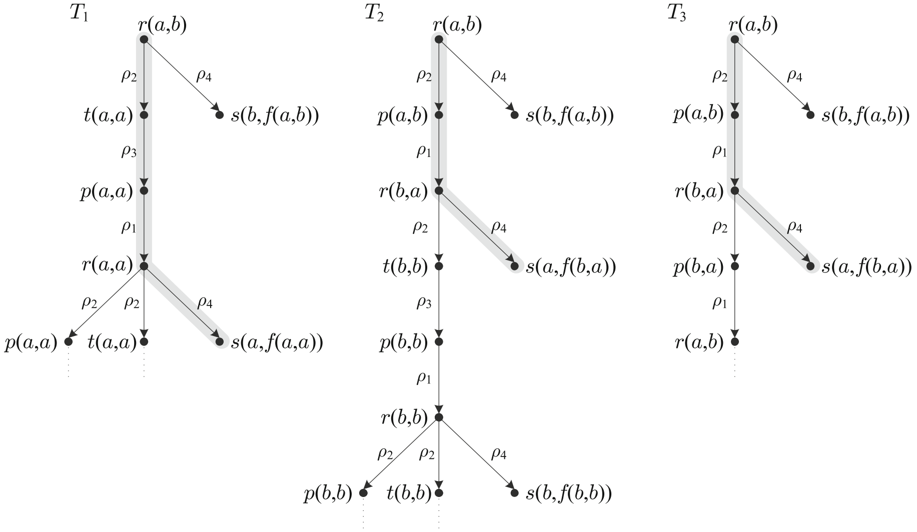


Fig. 2. Possible trees of $trees(\underline{d}, \Sigma_f)$

For each $i \in \{1,2\}$, if $h(\underline{b}_i) \in M$, then there is a $u \in N$ labeled by $h(\underline{b}_i)$, $e = (v,u)$ belongs to E, and $\lambda_2(e) = \rho$. Notice that, by construction, $\{\lambda_1(v)\}_{v \in N} \subseteq M$. The *$\rho$-tree* of M, denoted as ρ-$\mathit{tree}(M)$, is obtained from $\mathit{tree}(M)$ by keeping the root node v, each edge $e = (v,u)$ which is labeled by ρ, and the subtree rooted at u. Let $(\rho\text{-})\mathit{trees}(\underline{c},\Sigma_f) = \{(\rho\text{-})\mathit{tree}(M) \mid M \in \mathit{mods}(\{\underline{c}\},\Sigma_f)\}$. Notice that, by abuse of notation, given a (ρ-)tree T and an atom $\underline{a}$, we write $T \models \underline{a}$ if $\{\lambda_1(v)\}_{v \in N} \models \underline{a}$. It is not hard to see that $\{\underline{c}\} \cup \Sigma_f \models \underline{a}$ iff $T \models \underline{a}$, for each $T \in \mathit{trees}(\underline{c},\Sigma_f)$.

Example 2. Consider the set Σ_f of rules

$$\begin{array}{ll} \rho_1 : p(X,Y) \rightarrow r(Y,X) & \rho_2 : r(X,Y) \rightarrow p(X,Y) \vee t(X,X) \\ \rho_3 : t(X,Y) \rightarrow p(X,Y) & \rho_4 : r(X,Y) \rightarrow s(Y,f(X,Y)). \end{array}$$

Possible models of $\{r(a,b)\} \cup \Sigma_f$ are:

$$\begin{array}{l} M_1 = \{r(a,b), s(b,f(a,b)), t(a,a), p(a,a), r(a,a), s(a,f(a,a))\}, \\ M_2 = \{r(a,b), s(b,f(a,b)), p(a,b), r(b,a), t(b,b), p(b,b), r(b,b), s(a,f(b,a)), s(b,f(b,b))\}, \\ M_3 = \{r(a,b), s(b,f(a,b)), p(a,b), r(b,a), p(b,a), s(a,f(b,a))\}. \end{array}$$

Notice that, for each other model M of $\{r(a,b)\} \cup \Sigma_f$, it holds that $M_i \subseteq M$, for at least one $i \in [3]$; the tree T_i of M_i is depicted in Figure 2. Observe that each T_i has exactly one ρ_2-tree and one ρ_4-tree. Moreover, the shaded paths form (modulo null renaming) the proof-tree of $r(a,b)$ and Σ_f shown in Figure 1(a), which is valid w.r.t. $s(a,X)$. □

Observe that the first edge of each shaded path in Figure 2 is labeled by the same rule. As shown in the next technical lemma, this is a general property that holds whenever the given database and set of DTGDs entail the given atom.

Lemma 2. *Consider a normalized set Σ of linear DTGDs, an atom $\underline{c}$ with $\mathit{terms}(\underline{c}) \subset \Gamma_\Sigma$, and an atom $\underline{a}$ with $\mathit{terms}(\underline{a}) \subset \Gamma \cup \Gamma_V$. It holds that $\{c\} \cup \Sigma_f \models \underline{a}$ iff there exists $\rho \in \Sigma_f$ such that $T \models \underline{a}$, for each $T \in \rho$-$\mathit{trees}(\underline{c},\Sigma_f)$.*

Proof (sketch). By definition of the tree of a model, $\mathit{trees}(\underline{c},\Sigma_f)$ is isomorphic to the set $\mathcal{T} = \big\{\uplus_{i \in [|\Sigma|]} T_i \mid \langle T_1,\ldots,T_{|\Sigma|}\rangle \in \times_{\rho \in \Sigma_f} \rho\text{-}\mathit{trees}(\underline{c},\Sigma_f)\big\}$, where $\uplus_{i \in [|\Sigma|]} T_i$ is the rooted tree obtained from the disjoint union of $T_1,\ldots,T_{|\Sigma|}$ after merging the root nodes (see Figure 3). We are now ready to show our lemma. ($\Rightarrow$) Assume that for each $\rho \in \Sigma_f$ there exists $T \in \rho$-$\mathit{trees}(\underline{c},\Sigma_f)$ such that $T \not\models \underline{a}$. Therefore, there exists $T' \in \mathcal{T}$ such that $T' \not\models \underline{a}$. Clearly, there is $T'' \in \mathit{trees}(\underline{c},\Sigma_f)$ isomorphic to T'. Assuming that $T'' = \mathit{tree}(M)$, for some $M \in \mathit{mods}(\{\underline{c}\},\Sigma_f)$, $M \not\models \underline{a}$; thus, $\{\underline{c}\} \cup \Sigma_f \not\models \underline{a}$. ($\Leftarrow$) This direction follows immediately. □

Given a fact $\underline{d}$, a set Σ of linear DTGDs, and an atom $\underline{a}$, by applying recursively the property provided by Lemma 2, one can build a proof-tree of $\underline{d}$ and Σ which is valid w.r.t. $\underline{a}$. This is exactly the key idea underlying the proof of the next result.

Lemma 3. *Consider a database D, a normalized set Σ of linear DTGDs, and an atom $\underline{a}$ with $\mathit{terms}(\underline{a}) \subset \Gamma \cup \Gamma_V$. It holds that $D \cup \Sigma \models \underline{a}$ iff there exists a proof-tree of $\underline{d}$ and Σ, for some $\underline{d} \in D$, which is valid w.r.t. $\underline{a}$.*

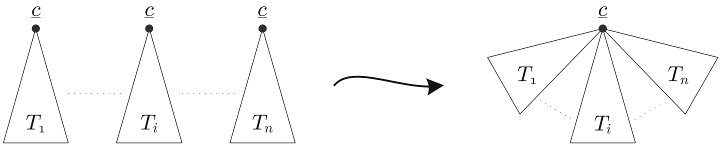


Fig. 3. Rooted tree obtained from the disjoint union of $T_1, \ldots, T_n$ after merging the root nodes

Proof (sketch). ($\Rightarrow$) The claim is shown by giving a proof-tree of some $\underline{d} \in D$ and Σ which is valid w.r.t. $\underline{a}$. By Lemmas 1 and 2, there are $\underline{d} \in D$ and $\rho \in \Sigma_f$ such that in every $T \in \rho\text{-}trees(\underline{d}, \Sigma_f)$ there is a path π_T from the root to a node u which is labeled by $h(\underline{a})$, where h is a homomorphism, and the first edge e_T of π_T is labeled by a rule ρ of the form $\underline{b} \rightarrow \underline{b}_1 \vee \underline{b}_2$ or $\underline{b} \rightarrow \underline{b}_1$. Let us now construct a rooted binary tree $T' = \langle N, E, \lambda_1, \lambda_2 \rangle$; initially, $N = \{v\}$, $E = \varnothing$, $\lambda_1 = \{v \rightarrow \underline{d}\}$, and $\lambda_2 = \varnothing$. Clearly, $\rho\text{-}trees(\underline{d}, \Sigma_f)$ can be partitioned into S_1 and S_2 such that, for each $T \in S_1$, if $e_T = (u, w)$ and h maps $\underline{b}$ into $\underline{d}$ (which is the label of u), then w is labeled by $h(\underline{b}_1)$. For each $i \in \{1, 2\}$, assume that the second node of π_T, for each $T \in S_i$, is labeled by $\underline{g}_i$. Let $N = N \cup \{w_1, w_2\}$, $E = E \cup \{(v, w_i)\}_{i \in \{1,2\}}$, $\lambda_1 = \lambda_1 \cup \{w_i \rightarrow \underline{g}_i\}_{i \in \{1,2\}}$, and $\lambda_2 = \lambda_2 \cup \{(v, w_i) \rightarrow \rho\}_{i \in \{1,2\}}$. Due to the fact that $\{\underline{g}_i\} \cup \Sigma_f \models \underline{a}$, for each $i \in [2]$, Lemma 2 can be recursively applied finitely many times as described above, and eventually T' is constructed. The desired proof-tree is obtained from T' by replacing the skolem terms occurring in T' with distinct nulls of Γ_N.

($\Leftarrow$) By hypothesis, there exists $\underline{d} \in D$ and a proof-tree T of $\underline{d}$ and Σ which is valid w.r.t. $\underline{a}$. It is possible to construct from T a rooted binary tree T' such that the nodes are labeled by atoms with skolem terms (instead of nulls), the edges are labeled by rules of Σ_f, and the structural properties of T are preserved. Assume that the label of each outgoing edge of the root of T' is ρ. By Lemma 1 and 2, it suffices to show that each ρ-tree of $\underline{d}$ and Σ_f entails $\underline{a}$. This follows from the fact that, for each $T'' \in \rho\text{-}trees(\underline{d}, \Sigma_f)$, there is a path from the root to a leaf of T' that can be mapped into T''. □

Interestingly, a fact $\underline{d}$ and a normalized set Σ of linear DTGDs admit a valid proof-tree w.r.t. an atom $\underline{a}$ iff they admit a "small" valid proof-tree T w.r.t. $\underline{a}$ involving at most $\omega + 1$ nulls, where ω is the maximum arity over all predicates of Σ. This holds since, by definition, T can be constructed in such a way that any pair of edges of the form (v, u) and (v, u') involves at most $\omega + 1$ nulls. By combining this observation with Lemma 3, we get that atom entailment is equivalent to the problem of deciding whether a "small" proof-tree exists. We do this by applying the alternating algorithm SearchPT given in Figure 4; an example of the computation of the algorithm follows.

Example 3. Consider the instance $I = \langle \underline{a}, \{r(a, b)\}, \Sigma \rangle$ of atom entailment, where Σ is:

$$\begin{array}{ll} \sigma_1 : r(X,Y) \rightarrow p(X) \vee t(Y,X) & \sigma_2 : p(X) \rightarrow r(X,X) \\ \sigma_3 : t(X,Y) \rightarrow \exists Z\, s(Y,Z) & \sigma_4 : s(X,Y) \rightarrow \exists Z\, s(Y,Z). \end{array}$$

Figure 5 shows an initial part of the alternating computation of SearchPT on I. Observe that in the shaded edge exactly three (maximum arity plus one) nulls appear. □

```
Algorithm SearchPT($\underline{a}, D, \Sigma$)

Input: An instance $\langle \underline{a}, D, \Sigma \rangle$ of ATOM-ENTAILMENT

1. $N := \{z_i \in \Gamma_N\}_{i \in [\omega+1]}$, and let $\rho : var(\underline{a}) \rightarrow N$ be a one-to-one substitution;
2. Existentially choose $\underline{c} \in D$;
3. Existentially choose to either execute step 4 or to skip to step 5;
4. If there exists a homomorphism $h$ such that $h(\underline{a}) = \underline{c}$, then accept;
5. Existentially choose $\sigma \in \Sigma$ and a homomorphism $h$ such that $h(body(\sigma)) = \underline{c}$; if there is
   no such a pair, then reject;
6. If $\sigma = p(\mathbf{X}, \mathbf{Y}) \rightarrow r_1(\mathbf{X}_1) \vee r_2(\mathbf{X}_2)$, then $\underline{b}_1 := h(r_1(\mathbf{X}_1))$ and $\underline{b}_2 := h(r_2(\mathbf{X}_2))$;
7. If $\sigma = p(\mathbf{X}, \mathbf{Y}) \rightarrow r(\mathbf{X})$, then $\underline{b}_1 := h(r(\mathbf{X}, Z))$ and $\underline{b}_2 := \rho(\underline{a})$;
8. If $\sigma = p(\mathbf{X}, \mathbf{Y}) \rightarrow \exists Z\, r(\mathbf{X}, Z)$, then $\underline{b}_1 := h'(r(\mathbf{X}, Z))$, where $h'$ is the homomorphism
   $h|_{\mathbf{X}} \cup \{Z \rightarrow t \mid t \in N,\ t \notin terms(h(p(\mathbf{X}, \mathbf{Y})))\}$, and $\underline{b}_2 := \rho(\underline{a})$;
9. Universally choose $\underline{c} \in \{\underline{b}_1, \underline{b}_2\}$ and goto step 3.
```

Fig. 4. The alternating algorithm SearchPT

Soundness and completeness of SearchPT follow by construction.

Proposition 1. *Given an instance $\langle \underline{a}, D, \Sigma \rangle$ of* ATOM-ENTAILMENT*, where Σ is a set of linear DTGDs, $D \cup \Sigma \models \underline{a}$ iff* SearchPT$(\underline{a}, D,$ Normalize$(\Sigma))$ *accepts.*

Equipped with the above machinery, we are now ready to establish the desired complexity upper bounds of our problem.

Theorem 2. ATOM-ENTAILMENT *under linear DTGDs is in **EXPTIME** in combined complexity, and in **PTIME** if the DTGDs have fixed size.*

Proof. Consider an instance $\langle \underline{a}, D, \Sigma \rangle$ of ATOM-ENTAILMENT, where Σ is a set of linear DTGDs. Since SearchPT is an alternating algorithm and $\Sigma' =$ Normalize(Σ) can be computed in polynomial time, to obtain the desired upper bounds, by Proposition 1, it suffices to show that SearchPT$(\underline{a}, D, \Sigma')$ runs in polynomial space in the general case, and in logarithmic space in the restricted case. At each step of the computation we need to remember at most two atoms involving ω terms, where each term can be represented using logarithmically many bits; more precisely, we need $\mathcal{O}(\omega \log \omega + \omega \log n)$ space, where $n = |terms(D)|$. Notice that, if the DTGDs of Σ have fixed size, then ω is also fixed (see Section 2), and the claim follows. □

The rest of this subsection is devoted to show that atom entailment under linear DTGDs is in $\mathbf{AC}_0$ in data complexity. We do this by establishing that linear DTGDs are first-order rewritable, i.e., atom entailment under linear DTGDs can be reduced to the problem of evaluating a first-order query over a database. First-order rewritability was first introduced in the context of description logics [18].

Consider a normalized set Σ of linear DTGDs over a schema $\mathcal{R}$, and an atom $\underline{a}$ over $\mathcal{R}$. Let C be the constants occurring in $\underline{a}$, and $N = \{z_1, \ldots, z_\omega\}$ be a set of nulls, where ω is the maximum arity over all predicates of $\mathcal{R}$. We call $base(\underline{a}, \mathcal{R})$ the

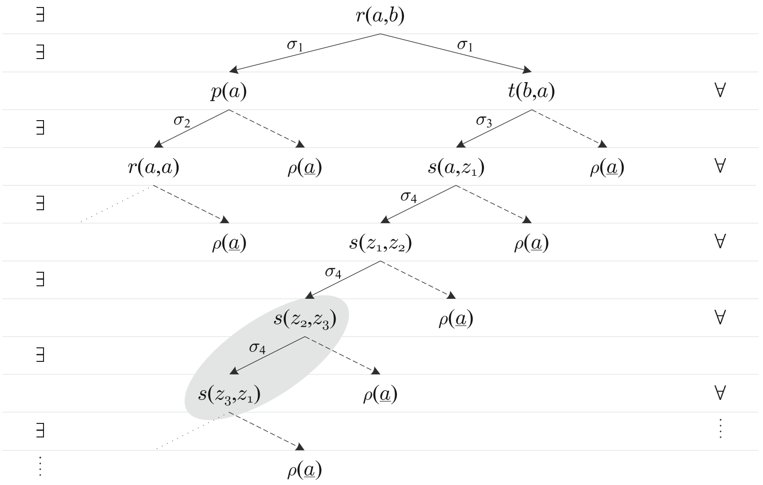


Fig. 5. Computation of the algorithm SearchPT

set of all atoms that can be formed with predicates of $\mathcal{R}$ and terms of $C \cup N$. Let $B = \{\underline{b} \mid \underline{b} \in \mathit{base}(\underline{a}, \mathcal{R}),\ \{\underline{b}\} \cup \Sigma \models \underline{a}\}$, and ρ be a renaming substitution that maps each $z \in N$ into a distinct variable $X_z \in \Gamma_V$. Let $Q[\underline{a}, \Sigma]$ be the first-order query $\exists X_{z_1} \ldots \exists X_{z_\omega} \bigvee_{\underline{b} \in B} \rho(\underline{b})$. The size of $Q[\underline{a}, \Sigma]$, defined as the number of disjuncts, is at most $|\mathcal{R}| \cdot (2\omega)^{\omega}$. Since, by Theorem 2, atom entailment under linear DTGDs is feasible in exponential time, the rewriting can be also constructed in exponential time. Notice that a database D entails $Q[\underline{a}, \Sigma]$, denoted $D \models Q[\underline{a}, \Sigma]$, if there exists a homomorphism h that maps $\rho(\underline{b})$ into D, for at least one atom $\underline{b} \in B$. In what follows, we show that $Q[\underline{a}, \Sigma]$ is a sound and complete rewriting of $\underline{a}$ and Σ.

Lemma 4. *Consider a database D, a normalized set Σ of linear DTGDs, and an atom $\underline{a}$ with $\mathit{terms}(\underline{a}) \subset \Gamma \cup \Gamma_V$. It holds that $D \cup \Sigma \models \underline{a}$ iff $D \models Q[\underline{a}, \Sigma]$.*

Proof (sketch). By construction of $Q[\underline{a}, \Sigma]$, it suffices to show that $D \cup \Sigma \models \underline{a}$ iff there exists $\underline{b} \in \mathit{base}(\underline{a}, \mathcal{R})$ and a homomorphism h such that $h(\underline{b}) \in D$ and $\{\underline{b}\} \cup \Sigma \models \underline{a}$. ($\Rightarrow$) By Lemma 1, there exists $\underline{d} \in D$ such that $\{\underline{d}\} \cup \Sigma \models \underline{a}$; therefore, there exists a homomorphism h_M that maps each $M \in \mathit{mods}(\{\underline{d}\}, \Sigma)$ into $\underline{a}$. Moreover, by construction, there exists $\underline{b} \in \mathit{base}(\underline{a}, \mathcal{R})$ and a bijective homomorphism μ such that $\mu(\underline{b}) = \underline{d}$; clearly, $\mu(\underline{b}) \in D$. Let $f : \mathit{terms}(\underline{d}) \rightarrow \mathit{terms}(\underline{b})$ be the substitution $\{t \rightarrow t' \mid t' \rightarrow t \in \mu|_{\mathit{terms}(\underline{d})}\}$. By induction on the depth of the trees of the models, it can be shown that for each $M' \in \mathit{mods}(\{\underline{b}\}, \Sigma)$, there exists $M \in \mathit{mods}(\{\underline{d}\}, \Sigma)$ such that $f(M) = M'$; thus, the homomorphism $f \circ h_M$ maps $\underline{a}$ into M'. ($\Leftarrow$) This direction can be easily shown by providing a similar argument. □

Since evaluation of first-order queries is feasible in $\mathbf{AC}_0$ [32], Lemma 4 implies the following complexity result.

Theorem 3. Atom-Entailment *under linear DTGDs is in $\mathbf{AC}_0$ in data complexity.*

4.2 Lower Bounds

We proceed now to establish the desired complexity lower bounds of atom entailment under linear DTGDs.

Theorem 4. *Consider an atom $\underline{a}$ over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of linear DTGDs over $\mathcal{R}$. The problem of deciding whether $D \cup \Sigma \models \underline{a}$ is* ***EXPTIME****-hard in combined complexity even if $|D| = 1$ and $|\mathcal{R}| = 2$.*

Proof (sketch). To prove our claim, it suffices to simulate the behavior of an **APSPACE** Turing machine M by means of linear DTGDs. W.l.o.g., we assume that M has exactly one accepting state s_{acc}, it is well-behaved and never "falls off" the left end of the tape, and also each configuration of M has at most two subsequent configurations. Moreover, we assume that the tape alphabet of M is $\{0, 1, \sqcup\}$, where $\sqcup$ denotes the blank symbol. Suppose that M halts on input $I = a_1 \ldots a_{|I|}$ using $n = |I|^k$ cells, where $k > 0$. Assume also that $s_1 < \ldots < s_m$, where $m > 0$, is the order that the states appear in the encoding of M, and that s_{init} is the initial state of M. We use a $(2n + m + 6)$-ary predicate called *config* to represent a configuration of M on I. An atom of the form $\mathit{config}(s, c_1, \ldots, c_n, b_1, \ldots, b_n, 0, 1, 0, 1, \sqcup, s_1, \ldots, s_m)$, where $s \in \{s_1, \ldots, s_m\}$, $\langle c_1, \ldots, c_n\rangle \in \{0, 1, \sqcup\}^n$, $\langle b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n\rangle = \{0\}^{n-1}$ and $b_i = 1$, represents the fact that M is in state s, the tape contains the string $c_1, \ldots, c_n$, and the cursor points at the i-th cell. The tuple $\langle 0, 1, 0, 1, \sqcup, s_1, \ldots, s_m\rangle$ (formally not part of the represented configuration) keeps the binary values 0 and 1, which will be used later in the simulation to move the cursor, the tape alphabet, and the states of the machine. Let D be the database that contains the atom

$$\mathit{config}(s_{init}, a_1, \ldots, a_{|I|}, \underbrace{\sqcup, \ldots, \sqcup}_{n-|I|}, 1, \underbrace{0, \ldots, 0}_{n-1}, 0, 1, 0, 1, \sqcup, s_1, \ldots, s_m)$$

which represents the initial configuration of M on I. We now encode the behavior of M on all transition rules that move the cursor to the right (resp., left) in the first (resp., second) subsequent configuration. For notational convenience, let

$$\mathbf{B}_\ell = \underbrace{Z_0, \ldots, Z_0}_{\ell-1}, Z_1, \underbrace{Z_0, \ldots, Z_0}_{n-\ell} \quad \text{and} \quad \mathbf{T} = Z_0, Z_1, V_0, V_1, V_\sqcup, S_1, \ldots, S_m,$$

where $\{Z_0, Z_1, V_0, V_1, V_\sqcup, S_1, \ldots, S_m\} \subset \Gamma_V$. For each transition rule of the form $\langle s_i, a\rangle \to \langle\langle s_j, b, \rightarrow\rangle, \langle s_k, c, \leftarrow\rangle\rangle$ in the transition function of M, we add in Σ the following linear DTGDs: for each $\ell \in [n-1]$, $\underline{a} \to \underline{b} \wedge \underline{c}$ if s_i is an $\exists$-state, and $\underline{a} \to \underline{b} \vee \underline{c}$ if s_i is a $\forall$-state, where

$$\begin{aligned}
\underline{a} &= \mathit{config}(S_i, X_1, \ldots, X_{\ell-1}, V_a, X_{\ell+1}, \ldots, X_n, \mathbf{B}_\ell, \mathbf{T}),\\
\underline{b} &= \mathit{config}(S_j, X_1, \ldots, X_{\ell-1}, V_b, X_{\ell+1}, \ldots, X_n, \mathbf{B}_{\ell+1}, \mathbf{T}),\\
\underline{c} &= \mathit{config}(S_k, X_1, \ldots, X_{\ell-1}, V_c, X_{\ell+1}, \ldots, X_n, \mathbf{B}_{\ell-1}, \mathbf{T}).
\end{aligned}$$

Similar DTGDs are used to encode the transition rules that move the cursor to the left or leave the cursor unmoved. Finally, assuming that $s_{acc} = s_i$, we add in Σ the following DTGDs: for each $\ell \in [n-1]$, $\mathit{config}(S_i, X_1, \ldots, X_n, \mathbf{B}_\ell, \mathbf{T}) \to \mathit{accept}(S_i)$. Notice that the above construction is feasible in polynomial time. It is not difficult to verify that, if $\underline{a} = accept(A)$, where $A \in \Gamma_V$, then M accepts I iff $D \cup \Sigma \models \underline{a}$. Since **APSPACE** coincides with **EXPTIME**, the claim follows from the fact that all the DTGDs of Σ are linear, the database D contains a single atom, and $\mathcal{R}$ contains only two predicates, namely, *config* and *accept*. □

It is interesting to say that in the above construction the arity of the predicate *config* can be reduced to $2n + 1$ if we allow the use of constants in the DTGDs. If we consider predicates of fixed arity, then the encoding given in the proof of Theorem 4 can be adapted in order to simulate an alternating logarithmic space Turing machine.

Theorem 5. *Consider an atom $\underline{a}$ over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of linear DTGDs over $\mathcal{R}$. Assuming that each DTGD of Σ has fixed size, the problem of deciding whether $D \cup \Sigma \models \underline{a}$ is **PTIME**-hard. The same lower bound holds even if $|D| = 1$ and each predicate of $\mathcal{R}$ is unary.*

Proof (sketch). It is well-known that **ALOGSPACE** equals **PTIME**. Thus, to prove our claim, it suffices to simulate the behavior of an **ALOGSPACE** Turing machine M on an input I by means of linear DTGDs with predicates of fixed arity. Recall that a logarithmic space Turing machine is equipped with a read-only input tape and a read/write work tape. Assume that M halts on $I = a_1, \ldots, a_{|I|}$ using $n = \log |I|^k$ cells, where $k > 0$. The key idea of the proof is to modify the construction given in the proof of Theorem 4 in such a way that a configuration of M on I is represented as a unary predicate. For example, the database atom can be encoded as

$$config[s_{init}\,\overbrace{a_1..a_{|I|}1\underbrace{0..0}_{|I|-1}}^{\text{input tape}}\overbrace{\underbrace{\sqcup..\sqcup}_{n}1\underbrace{0..0}_{n-1}}^{\text{work tape}}](c)$$

which represents the initial configuration of M on I, where c is an arbitrary constant of Γ. Since the number of configurations of M on I is polynomial, we need polynomially many unary predicates. The rest of the construction, which is feasible in logarithmic space, can be done by adapting the DTGDs given in the proof of Theorem 4. □

Acknowledgements. This research has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement DIADEM no. 246858, and from the Oxford Martin School's grant no. LC0910-019. Marco Manna's work was supported by the European Commission through the European Social Fund and by Calabria Region.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Alviano, M., Faber, W., Leone, N., Manna, M.: Disjunctive Datalog with existential quantifiers: Semantics, decidability, and complexity issues. In: TPLP (to appear, 2012)

3. Andréka, H., Németi, I., van Benthem, J.: Modal languages and bounded fragments of predicate logic. J. Philosophical Logic 27(3), 217–274 (1998)
4. Baader, F.: Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In: Proc. of IJCAI, pp. 319–324 (2003)
5. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of IJCAI, pp. 364–369 (2005)
6. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
7. Baget, J.-F., Leclère, M., Mugnier, M.-L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artif. Intell. 175(9-10), 1620–1654 (2011)
8. Bárány, V., Gottlob, G., Otto, M.: Querying the guarded fragment. In: Proc. of LICS, pp. 1–10 (2010)
9. Beeri, C., Vardi, M.Y.: The Implication Problem for Data Dependencies. In: Even, S., Kariv, O. (eds.) ICALP 1981. LNCS, vol. 115, pp. 73–85. Springer, Heidelberg (1981)
10. Berardi, D., Calvanese, D., Giacomo, G.D.: Reasoning on UML class diagrams. Artif. Intell. 168(1-2), 70–118 (2005)
11. Cabibbo, L.: The expressive power of stratified logic programs with value invention. Inf. Comput. 147(1), 22–56 (1998)
12. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. In: Proc. of KR, pp. 70–80 (2008)
13. Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies. In: Proc. of PODS, pp. 77–86 (2009)
14. Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., Pieris, A.: Datalog$^\pm$: A family of logical knowledge representation and query languages for new applications. In: Proc. of LICS, pp. 228–242 (2010)
15. Calì, A., Gottlob, G., Pieris, A.: Advanced processing for ontological queries. VLDB 3(1), 554–565 (2010)
16. Calì, A., Gottlob, G., Pieris, A.: Query Answering under Non-guarded Rules in Datalog$^\pm$. In: Hitzler, P., Lukasiewicz, T. (eds.) RR 2010. LNCS, vol. 6333, pp. 1–17. Springer, Heidelberg (2010)
17. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of KR, pp. 260–270 (2006)
18. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. Autom. Reasoning 39(3) (2007)
19. Chen, P.P.: The Entity-Relationship model - Toward a unified view of data. ACM Trans. Database Syst. 1(1), 9–36 (1976)
20. Deutsch, A., Nash, A., Remmel, J.B.: The chase revisited. In: Proc. of PODS, pp. 149–158 (2008)
21. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM Trans. Database Syst. 22(3), 364–418 (1997)
22. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theor. Comput. Sci. 336(1), 89–124 (2005)
23. Grädel, E.: On the restraining power of guards. J. Symb. Log. 64(4), 1719–1742 (1999)
24. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. Comput. Syst. Sci. 28(1), 167–189 (1984)
25. Krisnadhi, A., Lutz, C.: Data complexity in the EL family of DLs. In: Proc. of DL (2007)
26. Krötzsch, M., Rudolph, S.: Extending decidable existential rules by joining acyclicity and guardedness. In: Proc. of IJCAI, pp. 963–968 (2011)

27. Leone, N., Manna, M., Terracina, G., Veltri, P.: Efficiently computable Datalog$^{\exists}$ programs. In: Proc. of KR, pp. 13–23 (2012)
28. Marnette, B.: Generalized schema-mappings: From termination to tractability. In: Proc. of PODS, pp. 13–22 (2009)
29. Patel-Schneider, P.F., Horrocks, I.: A comparison of two modelling paradigms in the semantic web. J. Web Sem. 5(4), 240–250 (2007)
30. Poggi, A., Lembo, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. Data Semantics 10, 133–173 (2008)
31. Vardi, M.Y.: The complexity of relational query languages (extended abstract). In: Proc. of STOCS, pp. 137–146 (1982)
32. Vardi, M.Y.: On the complexity of bounded-variable queries. In: Proc. of PODS, pp. 266–276 (1995)

New Races in Parameterized Algorithmics

Christian Komusiewicz and Rolf Niedermeier

Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
{christian.komusiewicz,rolf.niedermeier}@tu-berlin.de

Abstract. Once having classified an NP-hard problem fixed-parameter tractable with respect to a certain parameter, the race for the most efficient fixed-parameter algorithm starts. Herein, the attention usually focuses on improving the running time factor exponential in the considered parameter, and, in case of kernelization algorithms, to improve the bound on the kernel size. Both from a practical as well as a theoretical point of view, however, there are further aspects of efficiency that deserve attention. We discuss several of these aspects and particularly focus on the search for "stronger parameterizations" in developing fixed-parameter algorithms.

1 Introduction

Efficiency is the central concern of algorithmics. In parameterized algorithmics, one tries to solve NP-hard problems in $f(k) \cdot n^{O(1)}$ time, where k is a problem-specific parameter (such as solution size), n is the size of the overall input, and f is an arbitrary function only depending (exponentially) on the parameter k [16, 23, 38]. Thus, the first step in parameterized algorithm design is to show that the considered problem is fixed-parameter tractable with respect to the chosen parameter k. Fixed-parameter algorithms are fast in case k is small and f grows "moderately". Consequently, once a problem is classified as fixed-parameter tractable, the race for the smallest function $f(k)$ starts.[1] There are many success stories in this direction, including problems such as Vertex Cover [13] and Undirected Feedback Vertex Set [11, 14]. Accompanied by these are similar races for the problem kernel size of the considered problem, for instance see the problems Cluster Editing [12] and again Undirected Feedback Vertex Set [45]. Furthermore, there is an ongoing deep theoretical effort for proving lower bounds (under complexity-theoretic assumptions) both for the $f(k)$ in the running time [24, 34] and the kernel size (polynomial vs non-polynomial) [8, 26].

In the two main lines of efficiency research in parameterized algorithmics described above, however, polynomial factors in the running time mostly are ignored. This is somewhat contrary to the fact that some key fixed-parameter tractability results have been termed "linear time for constant parameter value", basically meaning that the underlying problem can be solved in time $f(k) \cdot n$. Two

[1] See the web site on FPT races: http://fpt.wikidot.com/fpt-races

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 19–30, 2012.

examples in this direction are the "linear-time algorithms" for Treewidth [5] and Crossing Number [31]. As to striving for linear-time algorithms for effective data reduction, only recently the concept of "linear-time kernelization" gained more attention [3, 4, 29]. Altogether, these are important subjects of study besides the established races described in the beginning.

The focus of this article, however, is on another, practically and theoretically fruitful aspect in the race for efficiency. As an example, consider the following NP-hard problem with applications in graph drawing.

2-Layer Planarization
Input: An undirected graph $G = (V, E)$ and a $k \geq 0$.
Question: Is there an $E' \subseteq E$ with $|E'| \leq k$ such that deleting the edges in E' from G results in a biplanar graph.

Herein, a graph is biplanar if the vertices can be arranged on two parallel lines such that the edges (drawn as straight lines) do not cross. The best known fixed-parameter algorithm with respect to solution size k is due to Suderman [44] and runs in $O(3.6^k + |G|)$ time, improving on previous algorithms [18, 22] with exponential factors 6^k and 5.2^k, respectively. Later Uhlmann and Weller [47] followed a different improvement approach by considering the parameter "feedback edge set number" f. The essential point here is that for the parameter feedback edge set number f it holds that $f \leq k$ and f is expected to be significantly smaller than k in many realistic settings. In other words, f is a "stronger parameter" than k and thus the fixed-parameter tractability result with respect to the parameter f can be considered as an improvement over the fixed-parameter tractability result with respect to the solution size parameter k. Uhlmann and Weller [47] showed that 2-Layer Planarization can be solved in $O(6^f \cdot f^2 + f \cdot |E|)$ time.[2] If $1.7 \cdot f \leq k$, which can be the case in real-world instances, this algorithm is faster than the previous one.

In the spirit of the above considerations, we discuss in the following new efficiency races in parameterized algorithmics mainly based on the concept of stronger (and, correspondingly, weaker) parameterizations.

Let k_1 and k_2 be two natural numbers denoting parameters for input instances (here, graphs) of an NP-hard problem under study. Note that often, these parameters are functions of the input graph G (such as the maximum degree of G). We say that k_1 is *stronger than* k_2 if there is a constant c such that $k_1 \leq c \cdot k_2$ for all input instances of the underlying problem and furthermore there is *no constant* c' such that $k_2 \leq c' \cdot k_1$ in all input instances. Correspondingly, k_2 is *weaker than* k_1 in this case. There are several other reasonable possibilities to define the notions of weaker and stronger parameters. Here, we choose a "linear upper bound" for the following two reasons:

- For polynomial-size problem kernels, the main measure of effectiveness is the degree of the polynomial function in the size bound.

[2] Weller [48] recently reported on an improvement to $O(3.8^f \cdot f^2 + f \cdot |E|)$ time.

- In the theoretical analysis of running times of fixed-parameter algorithms, the most important feature is usually considered to be the function class of the exponential factor, for example $2^{O(k)}$ vs. $2^{O(k^2)}$.

Due to the linear bounds, these results can be transferred from stronger to weaker parameters. Note that having neither the stronger nor the weaker relation between two parameters does not necessarily imply that they are *incomparable*.

In the remainder of this article we describe some findings and challenges in the context of considering stronger and weaker parameters with respect to questions of efficiency. Herein, we deal both with structural parameters as well as parameterizations related to solution size.[3]

Preliminaries. We use n to denote the input size. A problem is called *fixed-parameter tractable* (FPT) if it can be solved in $f(k) \cdot \text{poly}(n)$ time, where f is a computable function only depending on k. The basic class of *parameterized intractability* is called W[1]. Problems that can be solved in polynomial time for constant parameter values are contained in the class XP. Note that these problems are not necessarily fixed-parameter tractable since the degree of the polynomial can be a function of the parameter k. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by data reduction. Here, the goal is for a given problem instance x with parameter k to transform it in polynomial time into a new instance x' with parameter $k' \leq k$ such that the size of x' is upper-bounded by some function g only depending on k and the instance (x, k) is a yes-instance if and only if (x', k') is a yes-instance. The reduced instance is called a *problem kernel*; in case the function g is a polynomial function it is called a *polynomial-size problem kernel. Turing kernelization* is a similar approach, where the main difference is that not only one, but polynomially many kernels can be created.

2 Structural Parameterizations or Navigating through Parameter Space

In this section, we give three examples for algorithmic studies of NP-hard graph problems that have been conducted in the spirit of identifying stronger or weaker parameters yielding tractability results. Most of the parameters considered in these studies are shown in Figure 1. The idea behind these parameters is that they measure the distance to easy, that is, polynomial-time solvable, input instances. Here, these are certain classes of graphs, for instance forests or bipartite graphs.

Small-Diameter Subgraphs. The following NP-hard problem is motivated by social and biological network analysis [1].

[3] While all our case studies are based on graph problems, the fundamental ideas and concepts presented here are clearly not restricted to these.

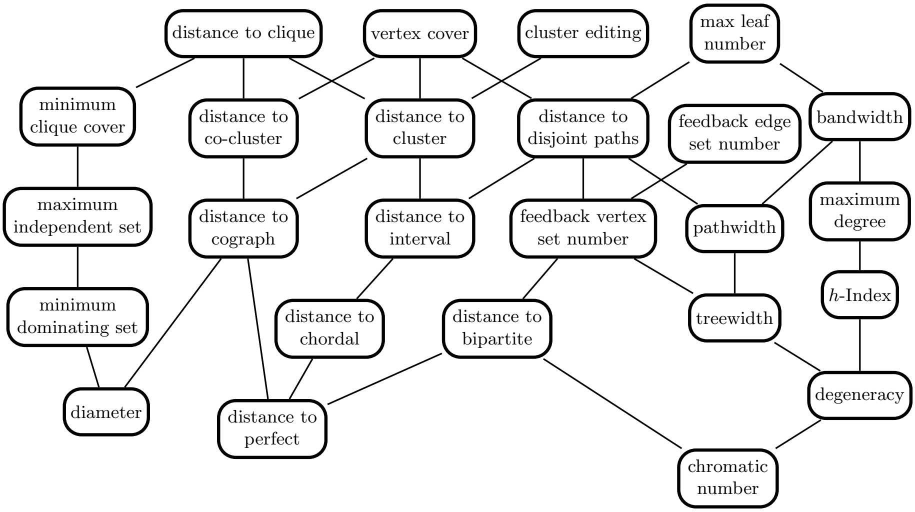


Fig. 1. An overview of the relation between some structural parameterizations for undirected graphs. Herein, "distance to X" is the number of vertices that have to be deleted in order to transform the input graph into a graph from the graph class X. For two parameters that are connected by a line, the upper parameter is weaker than the parameter below. For example, vertex cover is weaker than "distance to vertex-disjoint paths" since deleting a vertex cover produces an independent set, which also belongs to the graph class of disjoint paths. Similarly, distance to disjoint paths is weaker than "feedback vertex set number" which is weaker than "distance to bipartite".

> 2-Club
> Input: An undirected graph G and an integer ℓ.
> Question: Is there a subgraph G' with at least ℓ vertices that has diameter at most two?

By an easy polynomial-time data reduction rule it follows that 2-Club admits a Turing kernel with $O(\Delta^2)$ vertices where Δ is the *maximum degree* of G [42].[4] This implies fixed-parameter tractability for the parameter Δ. For applications in social network analysis, this parameterization seems not very useful, since social networks typically contain so-called *hubs*, that is, vertices of high degree. The number of hubs, however, is usually relatively small. Consequently, it is interesting to consider parameterizations expressing that many vertices in the graph have low degree. An established parameter in this context is the *degeneracy* of a graph: A graph G has degeneracy d if for each induced subgraph of G there is at least one vertex that has degree at most d. The concept of degeneracy was introduced as "coloring number" by Erdős and Hajnal [20].

While degeneracy is an interesting parameter, it is sometimes too strong in order to obtain fixed-parameter tractability results. An interesting alternative

[4] These results were presented for the parameter solution size ℓ. They also hold for the stronger parameter maximum degree Δ.

is therefore the *h-index* of a graph G. This is the maximum number h such that G contains h vertices of degree at least h [19]. Roughly speaking, the main difference between h-index and degeneracy is that a low h-index captures the global property that there are few high-degree vertices, whereas degeneracy can be one even if there are many high-degree vertices in the input graph. For example, a 2012 version of the DBLP co-author graph,[5] a typical social network, has $\approx 750,000$ vertices, maximum degree 804, h-index 208, and degeneracy 113. The 2-Club problem turns out to be solvable in $n^{f(h)}$ time but W[1]-hard with respect to the h-index of the input graph which also implies W[1]-hardness for the degeneracy of the input graph [30]. These two results now lead to the following two questions:

- Is 2-Club solvable in $n^{f(k)}$ time when k is the degeneracy of the input graph?
- Is there a parameter k between maximum degree and h-index for which 2-Club has the same kernelization properties as for the maximum degree, that is, it admits an $O(k^2)$-vertex Turing kernel?

Preprocessing for Treewidth. Computing the treewidth of a graph is a fundamental problem in graph algorithms:

Treewidth
Input: An undirected graph G, and an integer t.
Question: Does G have treewidth at most t?

Treewidth is fixed-parameter tractable with respect to the treewidth itself [5] but is unlikely to admit a polynomial-size problem kernel [8, 17]. However, several data reductions for the Treewidth problem are known to be effective in practice [6, 7]. In order to explain this effectiveness, the power of these data reduction rules was analyzed with respect to structural parameters that are weaker than treewidth [9]. On the one hand, it was shown that Treewidth admits an $O(k^3)$-vertex kernel when k is the vertex cover size, and an $O(k^4)$-vertex kernel when k is the size of a feedback vertex set of the input graph. On the other hand, it was shown that no polynomial-size kernels can be obtained for the distance to cluster graphs or the distance to chordal graphs [10]. Using the structural parameter map in Figure 1 one can now identify the parameters that are weaker or stronger than the previously considered parameters and thus identify the following open questions:

- Does Treewidth admit a problem kernel with $O(k^3)$ vertices when k denotes the "distance to disjoint paths"? As shown in Figure 1, this parameter is stronger than the vertex cover size and weaker than the feedback vertex set number.
- Is there a parameter between feedback vertex set size and treewidth, for example "distance to outerplanar graphs" for which Treewidth admits a polynomial-size problem kernel [9]?

[5] The graph was parsed using the data from http://dblp.uni-trier.de/xml/

Long Path and Long Cycle. In the NP-hard problems Long Path and Long Cycle one is given an undirected graph and asks for the existence of a simple path (or cycle, respectively) of length at least k. Both problems do not admit polynomial problem kernels with respect to the parameter solution size [8]. Motivated by this fact, Bodlaender et al. [10] studied both problems with respect to their kernelizability when parameterized by structural parameters. For instance, they showed that Long Cycle

- admits an $O(k^2)$-vertex problem kernel when parameterized by the vertex cover size,
- admits a polynomial problem kernel when parameterized by the size of the cluster vertex deletion set,
- does not admit a polynomial-size kernel when parameterized by the size of a vertex set whose deletion produces an outerplanar graph.

Since all forests are outerplanar, the parameter "distance to outerplanar graphs" is stronger than the parameter feedback vertex set number. Hence, Bodlaender et al. [10] posed the question whether Long Cycle admits a polynomial-size kernel when parameterized by feedback vertex set number. Assuming this question were settled, this would immediately raise another open question:

- In case Long Cycle admits a polynomial-size problem kernel when parameterized by the feedback vertex set number: is there a parameter that is stronger than feedback vertex set but for which Long Cycle still admits a polynomial-size problem kernel?
- In case Long Cycle does not admit a polynomial-size problem kernel when parameterized by the feedback vertex set number: is there a parameter stronger than vertex cover and weaker than feedback vertex set number, for which Long Cycle admits a polynomial kernel?

Furthermore, Bodlaender et al. [10] asked whether Long Cycle admits a polynomial-size problem kernel when parameterized by the "distance to co-graph". Similar to the discussion above, the answer to this question immediately raises a new question: either is there a parameter stronger than distance to co-graph such that Long Cycle admits a polynomial-size problem kernel, or is there a parameter weaker than distance to co-graph and stronger than cluster vertex deletion for which this is the case?

To conclude, navigating through the space of structural parameters (of which Figure 1 shows an excerpt) helps in identifying new research directions and open questions. Many of these questions are of purely algorithmic nature in the sense that the main question is fixed-parameter tractability (or the existence of a polynomial-size problem kernel) for a specific parameter. Some of these questions, however, also ask for the "discovery" of hidden parameters and are thus also closely related to general combinatorial aspects of the considered input structure.

3 Parameterizations Related to Solution Size

In this section, we present two examples for algorithmic studies in which parameterizations that are stronger than the "classical" parameterization by solution size have been proposed.

Above-Guarantee Parameterizations. Vertex Cover is one of the best-studied problems in parameterized algorithmics.

> Vertex Cover
> Input: An undirected graph G, and an integer k.
> Question: Is there a vertex set S of size at most k such that deleting S from G results in an independent set?

As mentioned in the introduction, the $f(k)$-race for Vertex Cover parameterized by vertex cover size k has brought a significant running time improvement. The vertex cover of a graph, however, is often relatively large. In order to obtain more meaningful parameters, Mahajan and Raman [35] proposed to study "parameterizations above guaranteed values". For vertex cover, one such parameterization is "parameterization above matching lower bound", that is, one asks whether there is a vertex cover of size at most $\ell + k$, where ℓ is the size of a maximum matching in the input graph. Clearly one vertex has to be used to cover each of the m matching edges and the parameter k measures the excess of this bound. The fixed-parameter tractability of Vertex Cover for this parameter follows from a fixed-parameter tractability result for Almost 2-SAT [41]. An $f(k)$-race for this parameter resulted in several improvements [15, 40].

Recently, Narayanaswamy et al. [36] showed that Vertex Cover is fixed-parameter tractable with respect to the "parameterization above linear program (LP) lower bound", that is, one asks whether there is a vertex cover of size at most $\ell + k$ where k is the value of an optimal solution of the LP relaxation of an ILP formulation of Vertex Cover. Again, the excess k above the lower bound is the parameter. Since the LP lower bound is at least as large as the matching lower bound, the new above-guarantee parameter is smaller. The fixed-parameter result by Narayanaswamy et al. [36] now starts a new $f(k)$-race for this parameter. For all three of the above-mentioned parameters there are algorithms that solve the problem in $c^k \cdot \text{poly}(n)$ time where c is a small constant. One interesting open question is thus: Is there an even stronger parameter for which a running time of $c^k \cdot \text{poly}(n)$ can be achieved, where c is relatively small.

Cluster Editing. Cluster Editing is a well-studied problem with applications in graph-based data clustering [28].

> Cluster Editing
> Input: An undirected graph G and an integer k.
> Question: Can G be transformed into a vertex-disjoint union of cliques (a so-called *cluster graph*) by at most k edge modifications?

Most investigations for Cluster Editing were concerned with the two classical FPT-races running time and problem kernel size for the parameter solution size k. More recently, other parameterizations for Cluster Editing have been considered. For instance, it was shown that Cluster Editing is fixed-parameter tractable when parameterized by the stronger parameter cluster vertex deletion number [32, 46] but NP-hard already on graphs of bounded degree [25, 33]. Furthermore, a parameter called "local modification bound" was introduced [33]. This parameter is a stronger parameter than the solution size: The solution size is the overall number of edge modifications; the local modification bound is the maximum of incident edge modifications over all vertices of the graph. Cluster Editing admits a kernel with at most $O(d \cdot t)$ vertices where d is an upper bound on the number of clusters in the cluster graph and t is the local modification bound [33] .

By means of a simple dynamic programming algorithm, the above problem kernel result implies an algorithm with running time $2^{O(dt)}$ for Cluster Editing. In contrast, for the parameter "solution size k and cluster number d", Cluster Editing can be solved in $2^{O(\sqrt{dk})} \cdot \text{poly}(n)$ time [25]. On the other hand, $2^{O(\sqrt{dt})} \cdot \text{poly}(n)$ time is unlikely to be achievable [33]. Hence, is there a parameter x between solution size k and local modification bound t for which an $2^{O(\sqrt{dx})} \cdot \text{poly}(n)$-time algorithm can be achieved? For instance, it would be interesting to consider the parameter number ℓ of edge deletions performed by a solution to Cluster Editing:

- Is Cluster Editing fixed-parameter tractable with respect to the number ℓ of edge deletions performed by a size-k solution?
- Does Cluster Editing admit a problem kernel that is polynomial in ℓ?
- Can Cluster Editing be solved in $2^{O(\ell)} \cdot \text{poly}(n)$ or in $2^{O(\sqrt{d\ell})} \cdot \text{poly}(n)$ time?

Summarizing, a very natural way of obtaining parameterizations that are stronger than the solution size is the approach of "parameterizing above guarantee" as discussed in the Vertex Cover example. The identification of stronger parameters, however, is not limited to this approach as demonstrated by Cluster Editing where simple combinatorial considerations have been used to identify parameters that are stronger than the solution size parameterization.

4 Conclusion

With the advent of parameterized algorithmics from single-parameter studies to multi-parameter studies investigating the relation between parameters becomes more and more important. In this line, studying the "stronger"-relation between parameters as proposed in this paper is a natural and fruitful undertaking, directly leading to numerous challenges on designing efficient and practically relevant fixed-parameter algorithms. This is encompassed by challenging combinatorial questions related to the "parameter space" (cf. Figure 1) associated with a specific problem. The study of the relationship between structural parameters measuring the input complexity should not be limited to graph problems. Indeed, obtaining analogous parameter spaces for "string parameters and problems" or "set system parameters and problems" is an open field.

The topic of alternative races in parameterized algorithmics is closely related to multivariate algorithm design and analysis [21, 39] in at least two ways. First, in both approaches it is essential that various parameters come into play. Second, when combining parameters as done in the multivariate design of fixed-parameter algorithms, then this usually happens in order to either improve the running time of a fixed-parameter tractability result towards practical relevance (by adding parameters one clearly further restricts the generality of the result, hopefully for the benefit of improved efficiency) or to turn W[1]-hardness with respect to a specific parameterization into fixed-parameter tractability by further adding one or more parameters. In this spirit, a parameter k_1 is likely to be stronger than a combined parameter (k_1, k_2), whatever k_2 is. So, parameter addition gives a way to generate weaker parameters. Let us describe one specific open questions in this context: It is open whether the NP-hard arc routing problem Rural Postman is fixed-parameter tractable with respect to the number of weakly connected components of the graph induced by the "required arcs" [27, 43]. As a step towards answering this long-standing open problem, it is interesting to identify weaker parameters making the problem fixed-parameter tractable.

To conclude, let us only briefly mention two further efficiency races in the context of parameterized algorithmics. First, there are many further problems that can be studied along the lines of this article. In particular, it is a challenge to extend the presented approaches for graph-theoretic problems to more complex real-world problems. For instance, the NP-hard Target Set Selection (modeling the spread of influence) is a prominent graph problem occurring in social networks. It is known to be W[1]-hard with respect to the parameters treewidth and also the weaker parameter feedback vertex set size [2] while it becomes fixed-parameter tractable for the still weaker parameter vertex cover number [37]. Second, a standard way of starting races is instead of making the parameter stronger is to make the underlying classes of input instances larger. For instance, not changing the parameter, does a result holding for planar graphs generalize to bounded-genus graphs and further to certain classes of minor-free graphs? This is closer to our approach than it might appear at first sight since many structural graph parameters basically also restrict the allowed graph types, thus also defining a specific graph class.

Finally, from a more applied point of view, the most natural way of spotting relevant parameterizations is to make measurements in the (real-world) input data. Quantities that turn out to be small particularly qualify for parameterized complexity studies. To this end, tools for data analysis are needed.[6] After having performed the data analysis task, one may set up the parameter navigation map and perform algorithmic studies as sketched in this article.

[6] We have implemented the *Graphana* tool, which can be used to compute or estimate graph parameters as shown in Figure 1. The software is available from `http://fpt.akt.tu-berlin.de/graphana`

Acknowledgments. We thank André Nichterlein, Manuel Sorge, and Mathias Weller for their comments which have improved this article.

References

[1] Balasundaram, B., Butenko, S., Trukhanovzu, S.: Novel approaches for analyzing biological networks. Journal of Combinatorial Optimization 10(1), 23–39 (2005)

[2] Ben-Zwi, O., Hermelin, D., Lokshtanov, D., Newman, I.: Treewidth governs the complexity of target set selection. Discrete Optimization 8(1), 87–96 (2011)

[3] van Bevern, R.: Towards Optimal and Expressive Kernelization for d-Hitting Set. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 121–132. Springer, Heidelberg (2012)

[4] van Bevern, R., Hartung, S., Kammer, F., Niedermeier, R., Weller, M.: Linear-Time Computation of a Linear Problem Kernel for Dominating Set on Planar Graphs. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 194–206. Springer, Heidelberg (2012)

[5] Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25(6), 1305–1317 (1996)

[6] Bodlaender, H.L., Koster, A.M.C.A.: Safe separators for treewidth. Discrete Mathematics 306(3), 337–350 (2006)

[7] Bodlaender, H.L., Koster, A.M.C.A., van den Eijkhof, F.: Preprocessing rules for triangulation of probabilistic networks. Computational Intelligence 21(3), 286–305 (2005)

[8] Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. Journal of Computer and System Sciences 75(8), 423–434 (2009)

[9] Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Preprocessing for Treewidth: A Combinatorial Analysis through Kernelization. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 437–448. Springer, Heidelberg (2011)

[10] Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Kernel Bounds for Path and Cycle Problems. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 145–158. Springer, Heidelberg (2012)

[11] Cao, Y., Chen, J., Liu, Y.: On Feedback Vertex Set New Measure and New Structures. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 93–104. Springer, Heidelberg (2010)

[12] Chen, J., Meng, J.: A $2k$ kernel for the cluster editing problem. Journal of Computer and System Sciences 78(1), 211–220 (2012)

[13] Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theoretical Computer Science 411(40-42), 3736–3756 (2010)

[14] Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011), pp. 150–159. IEEE (2011)

[15] Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J.O.: On Multiway Cut Parameterized above Lower Bounds. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 1–12. Springer, Heidelberg (2012)

[16] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)

[17] Drucker, A.: New limits to classical and quantum instance compression (2012) (manuscript, April 2012)

[18] Dujmovic, V., Fellows, M.R., Hallett, M.T., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F.A., Suderman, M., Whitesides, S., Wood, D.R.: A fixed-parameter approach to 2-layer planarization. Algorithmica 45(2), 159–182 (2006)

[19] Eppstein, D., Spiro, E.S.: The h-Index of a Graph and Its Application to Dynamic Subgraph Statistics. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 278–289. Springer, Heidelberg (2009)

[20] Erdős, P., Hajnal, A.: On chromatic number of graphs and set-systems. Acta Mathematica Academiae Scientiarum Hungaricae 17, 61–99 (1966)

[21] Fellows, M.: Towards Fully Multivariate Algorithmics: Some New Results and Directions in Parameter Ecology. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 2–10. Springer, Heidelberg (2009)

[22] Fernau, H.: Two-layer planarization: Improving on parameterized algorithmics. Journal of Graph Algorithms and Applications 9(2), 205–238 (2005)

[23] Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006)

[24] Flum, J., Grohe, M., Weyer, M.: Bounded fixed-parameter tractability and $\log^2 n$ nondeterministic bits. Journal of Computer and System Sciences 72(1), 34–71 (2006)

[25] Fomin, F.V., Kratsch, S., Pilipczuk, M., Pilipczuk, M., Villanger, Y.: Subexponential fixed-parameter tractability of cluster editing. CoRR, abs/1112.4419 (2011)

[26] Fortnow, L., Santhanam, R.: Infeasibility of instance compression and succinct PCPs for NP. Journal of Computer and System Sciences 77(1), 91–106 (2011)

[27] Frederickson, G.N.: Approximation algorithms for some postman problems. Journal of the ACM 26(3), 538–554 (1979)

[28] Guo, J.: Fixed-Parameter Algorithms for Graph-Modeled Date Clustering. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 39–48. Springer, Heidelberg (2009)

[29] Hagerup, T.: Simpler Linear-Time Kernelization for Planar Dominating Set. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 181–193. Springer, Heidelberg (2012)

[30] Hartung, S., Komusiewicz, C., Nichterlein, A.: On structural parameterizations for the 2-Club problem: Classical and parameterized hardness (2012) (manuscript, June 2012)

[31] Kawarabayashi, K., Reed, B.A.: Computing crossing number in linear time. In: Proceedings of the 39th ACM Symposium on Theory of Computing (STOC 2007), pp. 382–390. ACM (2007)

[32] Komusiewicz, C., Uhlmann, J.: Alternative Parameterizations for Cluster Editing. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Královič, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 344–355. Springer, Heidelberg (2011)

[33] Komusiewicz, C., Uhlmann, J.: Cluster editing with locally bounded modifications. Discrete Applied Mathematics (2012) (online available)

[34] Lokshtanov, D., Marx, D., Saurabh, S.: Lower bounds based on the exponential time hypothesis. Bulletin of the EATCS 105, 41–72 (2011)

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

[36] Narayanaswamy, N.S., Raman, V., Ramanujan, M.S., Saurabh, S.: LP can be a cure for parameterized problems. In: Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012). LIPIcs, vol. 14, pp. 338–349. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)

[37] Nichterlein, A., Niedermeier, R., Uhlmann, J., Weller, M.: On tractable cases of target set selection. Social Network Analysis and Mining (2012) (online available)

[38] Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications, vol. 31. Oxford University Press (2006)

[39] Niedermeier, R.: Reflections on multivariate algorithmics and problem parameterization. In: Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010). LIPIcs, vol. 5, pp. 17–32. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2010)

[40] Raman, V., Ramanujan, M.S., Saurabh, S.: Paths, Flowers and Vertex Cover. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 382–393. Springer, Heidelberg (2011)

[41] Razgon, I., O'Sullivan, B.: Almost 2-sat is fixed-parameter tractable. Journal of Computer and System Sciences 75(8), 435–450 (2009)

[42] Schäfer, A., Komusiewicz, C., Moser, H., Niedermeier, R.: Parameterized computational complexity of finding small-diameter subgraphs. Optimization Letters 6(5), 883–891 (2012)

[43] Sorge, M., van Bevern, R., Niedermeier, R., Weller, M.: A new view on rural postman based on Eulerian extension and matching. Journal of Discrete Algorithms (2012) (available online)

[44] Suderman, M.: Layered Graph Drawing. PhD thesis, School of Computer Science, McGill University (2005)

[45] Thomassé, S.: A $4k^2$ kernel for feedback vertex set. ACM Transactions on Algorithms 6(2) (2010)

[46] Uhlmann, J.: Multivariate Algorithmics in Biological Data Analysis. PhD thesis, Technische Universität Berlin, Berlin, Germany (2011)

[47] Uhlmann, J., Weller, M.: Two-Layer Planarization Parameterized by Feedback Edge Set. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 431–442. Springer, Heidelberg (2010)

[48] Weller, M.: An improved branching algorithm for two-layer planarization parameterized by the feedback edge set number (manuscript, 2012)

Scott Is Always Simple

Antonino Salibra

DAIS, Università Ca'Foscari Venezia
Via Torino 155, 30172 Venezia, Italy
salibra@dsi.unive.it

Abstract. In this paper we give an outline of recent algebraic results concerning theories and models of the untyped lambda calculus.

1 Introduction

The lambda calculus was originally introduced by Church [12,13] as a foundation for logic, where functions, instead of sets, were primitive, and it turned out to be consistent and successful as a tool for formalising all computable functions. The rise of computers and the development of programming languages gave a new development to its theoretical studies. The lambda calculus is the kernel of the functional programming paradigm, because its ordinary parameter-binding mechanism corresponds closely to parameter binding in many functional programming languages and to variable binding of quantifiers in logic.

Lambda calculus has been originally investigated by using mainly syntactical methods (see Barendregt's book [1]). At the beginning researchers have focused their interest on a limited number of equational extensions of lambda calculus, called λ-*theories*. They arise by syntactical or semantic considerations. Indeed, a λ-theory may correspond to a possible operational semantics of lambda calculus, as well as it may be induced by a model of lambda calculus through the kernel congruence relation of the interpretation function. The set of λ-theories is naturally equipped with a structure of complete lattice (see [1, Chapter 4]). The bottom element of this lattice is the least λ-theory $\lambda\beta$, while the top element is the inconsistent λ-theory. Although researchers have mainly focused their interest on a limited number of them, the lattice of λ-theories has a very rich and complex structure (see e.g. [1,19,21]).

The lambda calculus, although its axioms are all in the form of equations, is not a genuine equational theory since the variable-binding properties of lambda abstraction prevent "variables" in lambda calculus from operating as real algebraic variables. There have been several attempts to reformulate the lambda calculus as a purely algebraic theory. The earliest, and best known, algebraic models are the combinatory algebras of Curry and Schönfinkel (see [15]). Although combinatory algebras do not keep the lambda notation, they have a simple purely equational characterisation and were used to provide an intrinsic first-order, but not equational, characterisation of the models of lambda calculus, as a special class of combinatory algebras called λ-*models* [1, Def. 5.2.7].

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 31–45, 2012.

The connection between the syntax and the semantics of lambda calculus is established by the completeness theorem of lambda calculus: every λ-theory is the equational theory of some λ-model.

Semantical methods have been extensively investigated. After the first model, found by Scott [28] in 1969 in the category of complete lattices and Scott continuous functions, a large number of mathematical models for lambda calculus have been introduced in various categories of domains and were classified into semantics according to the nature of their representable functions, see e.g. [1,3]. Scott continuous semantics [29] is given in the category whose objects are complete partial orders and morphisms are Scott continuous functions. Other semantics of lambda calculus were isolated by Berry [5] and Bucciarelli-Ehrhard [7]: Berry's stable semantics and Bucciarelli-Ehrhard's strongly stable semantics are refinements of the continuous semantics introduced to capture the notion of "sequential" Scott continuous function. All these semantics are structurally and equationally rich in the sense that it is possible to build up $2^{\aleph_0}$ λ-models in each of them inducing pairwise distinct λ-theories (see [17,18]). Nevertheless, the above denotational semantics do not match all possible operational semantics of lambda calculus. We recall that a semantics of lambda calculus is equationally incomplete if there exists a λ-theory which is not the theory of any model in the semantics. In the nineties the problem of the equational incompleteness was positively solved by Honsell and Ronchi della Rocca [16] for Scott's continuous semantics, and by Bastonero and Gouy for Berry's stable semantics [2]. The proofs of the above results are syntactical and very difficult. In [26] the author has provided an algebraic and simple proof of the equational incompleteness of all semantics of lambda calculus that involve monotonicity with respect to some partial order and have a bottom element (including the incompleteness of the strongly stable semantics, which had been conjectured by Bastonero-Gouy and by Berline [2,3]).

The need of more abstract and sophisticated mathematical techniques in lambda calculus arises when we recognise the difficulty of the problems we handle, for example in order to investigate the structure of the lattice of λ-theories in itself and in connections with the theory of models. The author [19,25,26] has launched at the end of the nineties a research program for exploring lambda calculus and combinatory logic using techniques of universal algebra. The remark that the lattice of λ-theories is isomorphic to the congruence lattice of the term algebra of the least λ-theory $\lambda\beta$ is the starting point for studying lambda calculus by universal algebraic methods, through the variety (i.e., equational class) generated by the term algebra of $\lambda\beta$. In [25] the author has shown that the variety generated by the term algebra of $\lambda\beta$ is axiomatised by the finite schema of identities characterising *λ-abstraction algebras* (see [23]). The variety of λ-abstraction algebras is an algebraic description of lambda calculus, which keeps the lambda notation and hence all the functional intuitions. In [25] it was shown that, for every variety of λ-abstraction algebras, there exists exactly one λ-theory whose term algebra generates the variety. Thus, the properties of a λ-theory can be studied by means of the variety of λ-abstraction algebras generated by its term algebra. By applying these methods it was shown in [19] that the lattice of λ-theories satisfies a nontrivial implication in the language of lattices. It is open whether the lattice of λ-theories satisfies a nontrivial lattice identity (see [21, Section 4] for other results concerning the structure of the lattice of λ-theories).

Longstanding open problems of lambda calculus can be restated in terms of algebraic properties of varieties of λ-abstraction algebras or combinatory algebras. For example, the open problem of the order-incompleteness of lambda calculus, raised by Selinger (see [30]), asks for the existence of a λ-theory not arising as the equational theory of a non-trivially partially ordered model of lambda calculus. A partial answer to the order-incompleteness problem was obtained by the author in [26], where it is shown the existence of a λ-theory not arising as the equational theory of a non-trivially partially ordered model with a *finite* number of connected components. The order-incompleteness of lambda calculus is equivalent to the existence of an n-permutable variety of combinatory algebras for some natural number $n \geq 2$ (see the remark after Thm. 3.4 in [30]). Plotkin, Selinger and Simpson (see [30]) have shown that 2-permutability and 3-permutability are inconsistent with lambda calculus. The problem of n-permutability remains open for $n \geq 4$.

One of the milestones of modern algebra is the Stone representation theorem for Boolean algebras. This result was first generalised by Pierce [22] to commutative rings with unit and next by Comer [14] to the class of algebras with Boolean factor congruences. Comer's generalisation of Stone representation theorem also holds for combinatory algebras (see [21]): any combinatory algebra is isomorphic to a weak Boolean product of directly indecomposable algebras (i.e., algebras which cannot be decomposed as the Cartesian product of two other non-trivial algebras). The proof of the representation theorem is based on the fact that the directly indecomposable combinatory algebras constitute a universal class and that every combinatory algebra contains a Boolean algebra of central elements (introduced by Vaggione [31] in universal algebra). These elements define a direct decomposition of the algebra as the Cartesian product of two other algebras, just like idempotent elements in rings. This approach to central elements can be developed in the more general context of Church algebras, which were introduced in [20] to equationally axiomatise the "if-then-else" construct of programming.

The Stone representation theorem can be roughly summarised as follows: the directly indecomposable combinatory algebras are the 'building blocks' of the variety of combinatory algebras. The notion of directly indecomposable combinatory algebra appears to be so relevant that it is even interesting to speak of the "*indecomposable semantics*" to denote the class of models of λ-calculus which are directly indecomposable as combinatory algebras. This semantics encompasses the Scott continuous, stable and strongly stable semantics and was shown incomplete in [21].

The paper is organised as follows. In Section 2 we review the basic definition of lambda calculus and universal algebra. Church algebras are presented in

Section 3. We provide the algebraic incompleteness theorem in Section 4. The last section is devoted to the order-incompleteness problem of λ-calculus.

2 Preliminaries

2.1 Lambda Calculus

With regard to the λ-calculus we follow the notation and terminology of [1]. Λ and Λ^o are, respectively, the set of λ-terms and of closed λ-terms. Ω denotes the looping term $(\lambda x.xx)(\lambda x.xx)$.

We denote $\alpha\beta$-conversion by $\lambda\beta$ and $\alpha\beta\eta$-conversion by $\lambda\beta\eta$. A *λ-theory* is a congruence on Λ (with respect to the operators of abstraction and application) which contains $\lambda\beta$. A λ-theory is *consistent* if it does not equate all λ-terms, *inconsistent* otherwise. The set of λ-theories constitutes a complete lattice w.r.t. intersection, whose top is the inconsistent λ-theory and whose bottom is the theory $\lambda\beta$. The λ-theory generated by a set E of identities between λ-terms is the intersection of all λ-theories containing E.

There are two descriptions of what a model of λ-calculus is: the category-theoretical and the algebraic one. The categorical notion of model is well-suited for constructing concrete models, while the algebraic one is rather used to understand global properties of models (constructions of new models out of existing ones, closure properties, etc.) and to obtain results about the structure of the lattice of λ-theories. In the remaining part of this section we define the algebraic notion of model.

Definition 1. *An algebra* $\mathbf{C} = (C, \cdot, \mathbf{k}, \mathbf{s})$*, where* $\cdot$ *is a binary operation and* $\mathbf{k}, \mathbf{s}$ *are constants, is a* combinatory algebra *if it satisfies the identities* $\mathbf{k}xy = x$ *and* $\mathbf{s}xyz = xz(yz)$ *(as usual, the symbol "$\cdot$" is omitted and association is made on the left).*

In the equational language of combinatory algebras we define $\mathbf{i}$, $\mathbf{1}$ and $\mathbf{1}_n$ as follows: $\mathbf{i} \equiv \mathbf{skk}$; $\mathbf{1} \equiv \mathbf{1}_1 \equiv \mathbf{s}(\mathbf{ki})$ and $\mathbf{1}_{n+1} \equiv \mathbf{s}(\mathbf{k1})(\mathbf{s}(\mathbf{k1}_n))$. Hence, every combinatory algebra satisfies $\mathbf{i}x = x$ and $\mathbf{1}_n x_1 \dots x_n = x_1 \dots x_n$.

A function $f : C \to C$ is *representable* in a combinatory algebra $\mathbf{C}$ if there exists an element $c \in C$ such that $cz = f(z)$ for all $z \in C$.

Two elements $x, y \in C$ are called *extensionally equal* if they represent the same function in $\mathbf{C}$. For example, the elements x and $\mathbf{1}x$ are extensionally equal.

An *environment* is a function $\rho : Var \to C$, where Var is the set of variables of λ-calculus. For every variable x and $a \in C$ we denote by $\rho[x := a]$ the environment ρ' which coincides with ρ, except on x, where ρ' takes the value a.

Given a combinatory algebra $\mathbf{C}$, the interpretation of a λ-term M is defined by induction as follows, for every environment ρ:

$$|x|_\rho = \rho(x); \qquad |MN|_\rho = |M|_\rho \cdot |N|_\rho; \qquad |\lambda x.M|_\rho = \mathbf{1}m,$$

where $m \in C$ is any element representing the function $a \in C \mapsto |M|_{\rho[x:=a]}$. The drawback of the previous definition is that it may happen that the function $a \in$

$C \mapsto |M|_{\rho[x:=a]}$ is not representable in $\mathbf{C}$. The axioms characterising λ-models were expressly chosen to make coherent the previous definition of interpretation.

Definition 2. *A combinatory algebra* $\mathbf{C}$ *is called a* λ-model *if it satisfies the identities* $\mathbf{1}_2\mathbf{k} = \mathbf{k}$*,* $\mathbf{1}_3\mathbf{s} = \mathbf{s}$ *and the Meyer-Scott axiom:*

$$\forall x \forall y (\forall z (xz = yz) \Rightarrow \mathbf{1}x = \mathbf{1}y).$$

Here the combinator $\mathbf{1}$ is used as an inner choice operator. Indeed, given any $x \in C$, the element $\mathbf{1}x$ is in the same equivalence class as x w.r.t. extensional equality; and, by Meyer-Scott axiom, $\mathbf{1}x = \mathbf{1}y$ for every y extensionally equal to x. Thus, the set Y of elements representing the function $a \in C \mapsto |M|_{\rho[x:=a]}$ admits $\mathbf{1}m$ as a canonical representative and this does not depend on the choice of $m \in Y$.

We write $\mathbf{C} \models M = N$ if $|M|_\rho = |N|_\rho$ for all environments ρ. A λ-model univocally induces a λ-theory through the kernel congruence relation of the interpretation function:

$$Th(\mathbf{C}) = \{(M, N) \in \Lambda \times \Lambda : \mathbf{C} \models M = N\}.$$

If T is a λ-theory, the (open) *term model* of T is the algebra $(\Lambda/T, \cdot, \mathbf{k}, \mathbf{s})$, where $t/T \cdot u/T = (tu)/T$ and $\mathbf{k}$, $\mathbf{s}$ are respectively the equivalence classes of $\lambda xy.x$ and $\lambda xyz.xz(yz)$ modulo T.

A *partially ordered* λ*-model*, a *po-model* for short, is a pair $(\mathbf{A}, \leq)$, where $\mathbf{A}$ is a λ-model and $\leq$ is a partial order on A which makes the application operator of $\mathbf{A}$ monotone in both arguments. A po-model $(\mathbf{A}, \leq)$ is *non-trivial* if the partial order is not discrete, i.e., $a < b$ for some $a, b \in A$ (thus A is not a singleton).

2.2 Algebras

With regard to Universal Algebra we follow the notation and terminology of [9]. A *type* ν is a set of operation symbols of finite arity. An *algebra* $\mathbf{A}$ *of type* ν is a tuple $(A, \sigma^{\mathbf{A}})_{\sigma \in \nu}$, where, for every $\sigma \in \nu$ of arity n, $\sigma^{\mathbf{A}}$ is a function from A^n into A.

Given two algebras $\mathbf{A}$ and $\mathbf{B}$ of type ν, a *homomorphism* from $\mathbf{A}$ into $\mathbf{B}$ is a map $g : A \to B$ such that $g(\sigma^{\mathbf{A}}(a_1, \dots, a_n)) = \sigma^{\mathbf{B}}(g(a_1), \dots, g(a_n))$ for each n-ary operation $\sigma \in \nu$ and for all $a_i \in A$. Two algebras $\mathbf{A}$ and $\mathbf{B}$ are *isomorphic*, and we write $\mathbf{A} \cong \mathbf{B}$, if there exists a bijective homomorphism from $\mathbf{A}$ into $\mathbf{B}$.

Given an algebra $\mathbf{A}$ of type ν, a binary relation ϕ on $\mathbf{A}$ is *compatible* if for all $\sigma \in \nu$ of arity n, and for all $a_i, b_i \in A$ we have

$$a_1 \phi b_1, \dots, a_n \phi b_n \to f^{\mathbf{A}}(a_1, \dots, a_n) \phi f^{\mathbf{A}}(b_1, \dots, b_n).$$

A compatible equivalence relation is called a *congruence.*

The *kernel* of a homomorphism $g : \mathbf{A} \to \mathbf{B}$ is the congruence $ker(g) = \{(a, b) \in A^2 : g(a) = g(b)\}$.

Given two congruences ϕ and ψ, we can form their *relative product*: $\psi \circ \phi = \{(a,c) : (\exists b \in A)\ a\phi b\psi c\}$. It is easy to check that $\psi \circ \phi$ is still a compatible relation, but not necessarily a congruence.

We denote by $Con(\mathbf{A})$ the complete lattice of the congruences of $\mathbf{A}$, which is a sublattice of the lattice of the equivalence relations on $\mathbf{A}$. The meet $\phi \wedge \psi$ of two congruences ϕ and ψ is their intersection, while their join is the least equivalence relation including $\phi \cup \psi$:

$$\phi \vee \psi = \bigcup_{n>0} \phi \circ^n \psi,$$

where $\psi \circ^1 \phi = \psi$ and $\psi \circ^{n+1} \phi = \psi \circ (\phi \circ^n \psi)$ for $n > 0$. The diagonal $\Delta_{\mathbf{A}} = \{(a,a) : a \in A\}$ is the bottom element of $Con(\mathbf{A})$, while $\nabla_{\mathbf{A}} = A \times A$ is the top element.

$\mathbf{F}_\nu(X)$ is the *absolutely free algebra of type ν over a set X of generators.* The elements of $\mathbf{F}_\nu(X)$ are called terms and are built up by induction: (i) every element of X is a term; (ii) if $\sigma \in \nu$ is an operation symbol of arity n and $p_1, \dots, p_n$ are terms, then $\sigma(p_1, \dots, p_n)$ is a term. The operations of $\mathbf{F}_\nu(X)$ are the syntactical operations of term construction.

Hereafter, we omit the index ν because all algebras will be always of type ν.

If $\mathbf{A}$ is an algebra, then, for every map $\rho : X \to A$, there exists a unique homomorphism $\rho^* : \mathbf{F}(X) \to \mathbf{A}$ extending ρ: $\rho^*(x) = \rho(x)$ for every $x \in X$; $\rho^*(\sigma(p_1, \dots, p_n)) = \sigma^{\mathbf{A}}(\rho^*(p_1), \dots, \rho^*(p_n))$.

For every term p, we define $p^{\mathbf{A}} : A^X \to A$ the map defined by $p^{\mathbf{A}}(\rho) = \rho^*(p)$. When $X = \{x_1, \dots, x_n\}$ is a finite set of cardinality n, we define $p^{\mathbf{A}} : A^n \to A$ in the following equivalent way:

$$p^{\mathbf{A}}(a_1, \dots, a_n) = \rho^*(p), \text{ where } \rho(x_i) = a_i \text{ for } i = 1, \dots, n.$$

$p^{\mathbf{A}}$ is called a *term operation* of $\mathbf{A}$.

An algebra $\mathbf{A}$ satisfies an equation $p = q$ $(p, q \in F(X))$ if $p^{\mathbf{A}} = q^{\mathbf{A}}$. We write $\mathbf{A} \models p = q$ if $\mathbf{A}$ satisfies the equation $p = q$.

If $\mathbf{A}$ is an algebra, we denote by $Eq_X(\mathbf{A})$ ($Eq(\mathbf{A})$ for short) the set of equations $p = q$ $(p, q \in F(X))$ satisfied by $\mathbf{A}$.

If $\mathcal{K}$ is a class of algebras, then $Eq(\mathcal{K}) = \{p = q : (\forall \mathbf{A} \in \mathcal{K})\ \mathbf{A} \models p = q\}$ is the set of equations satisfied by every algebra of $\mathcal{K}$.

A class $\mathcal{K}$ of algebras is (i) *equational* if it is axiomatised by a set of equations; (ii) a *variety* if it is closed under Cartesian products, subalgebras and homomorphic images. A class of algebras is equational iff it is a variety.

2.3 The Building Blocks of Algebras

Direct Products. The Cartesian product $\mathbf{B} \times \mathbf{C}$ of two algebras $\mathbf{B}$ and $\mathbf{C}$ of the same type has $B \times C$ as universe and operations defined as follows, for every $a_i \in B$ and $b_i \in C$:

$$\sigma^{\mathbf{B}\times\mathbf{C}}(\langle a_1, b_1\rangle, \dots, \langle a_n, b_n\rangle) = \langle \sigma^{\mathbf{B}}(a_1, \dots, a_n), \sigma^{\mathbf{C}}(b_1, \dots, b_n)\rangle.$$

Lemma 1. *An algebra* $\mathbf{A}$ *is isomorphic to the Cartesian product* $\mathbf{B} \times \mathbf{C}$ *iff there exist two congruences* $\theta, \overline{\theta} \in \mathrm{C}on(\mathbf{A})$ *such that* $\mathbf{B} \cong \mathbf{A}/\theta$, $\mathbf{C} \cong \mathbf{A}/\overline{\theta}$, $\theta \cap \overline{\theta} = \Delta_{\mathbf{A}}$ *and* $\theta \circ \overline{\theta} = \nabla_{\mathbf{A}}$.

Definition 3. *A pair of congruences* $\langle \theta, \overline{\theta} \rangle$ *is called* a pair of complementary factor congruences *if* $\theta \cap \overline{\theta} = \Delta$, $\theta \circ \overline{\theta} = \nabla$. *A pair* $\langle \theta, \overline{\theta} \rangle$ *is trivial if either* $\theta = \Delta$ *or* $\overline{\theta} = \Delta$.

A congruence θ is called a *factor congruence* if there exists $\overline{\theta}$ such that $(\theta, \overline{\theta})$ is a pair of complementary factor congruences. We denote by $FC(\mathbf{A})$ the set of factor congruence of $\mathbf{A}$. In general, $FC(\mathbf{A})$ is not a sublattice of $\mathrm{C}on(\mathbf{A})$.

Definition 4. *An algebra* $\mathbf{A}$ *is* directly indecomposable *if it admits only the trivial pair of complementary factor congruences.*

In other words, $\mathbf{A}$ is directly indecomposable if, and only if, it cannot be decomposed as a nontrivial Cartesian product. Then the directly indecomposable algebras are the building blocks for products.

Factors Congruences and Decomposition Operators. Let $(\theta, \overline{\theta})$ be a pair of complementary factor congruences. Define a function $f_\theta : A \times A \to A$ as follows:

$$f_\theta(a, b) = \text{ the unique } c \text{ such that } a\theta c\overline{\theta}b.$$

Proposition 1. *The function* f_θ *satisfies the following conditions (for short, we write* f *for* f_θ*):*

(D1) $f(x, x) = x$;
(D2) $f(x, f(y, z)) = f(x, z) = f(f(x, y), z)$;
(D3) f *is a homomorphism from* $\mathbf{A} \times \mathbf{A}$ *into* $\mathbf{A}$.

Definition 5. *A function* $f : A \times A \to A$ *satisfying conditions (D1)-(D3) is called a* decomposition operator.

Proposition 2. *Given a decomposition operator* $h : A \times A \to A$, *the pair* $(\theta_h, \overline{\theta}_h)$, *defined by* $a\theta_h b \;\Leftrightarrow\; h(a, b) = a$; $\;a\overline{\theta}_h b \;\Leftrightarrow\; h(a, b) = b$, *is a pair of complementary factor congruences.*

In conclusion, we have:

Proposition 3. *There is a bijective correspondence between decomposition operators and factor congruences in such a way that*

$$\theta \mapsto f_\theta \mapsto \theta_{f_\theta} = \theta; \qquad\qquad h \mapsto \theta_h \mapsto f_{\theta_h} = h.$$

There is always a battle to simplify complex objects. Decomposition operators are more suitable to be internalized within an algebra than pairs of complementary factor congruences. We are particularly interested in algebras, whose decomposition operators are term operations of the algebra.

Subdirect and Boolean Products. If $\mathbf{A}$ and $\mathbf{B}$ are algebras, then we write $\mathbf{A} \leq \mathbf{B}$ if there is an embedding (= injective homomorphism) from $\mathbf{A}$ into $\mathbf{B}$.

Definition 6. *An algebra $\mathbf{A}$ is a subdirect product of an indexed family $(\mathbf{A}_i)_{i\in I}$ of algebras if $\mathbf{A} \leq \Pi_{i\in I}\mathbf{A}_i$ and $\pi_i(A) = A_i$ (where $\pi_i : A \to A_i$ is the projection in the i-coordinate).*

A direct product is an example of subdirect product.

Lemma 2. *Let $\theta_i \in \mathrm{C}on(\mathbf{A})$ be a family of congruences of an algebra $\mathbf{A}$. $\mathbf{A}$ is a subdirect product of the family $(\mathbf{A}/\theta_i)_{i\in I}$ (through the embedding $f(a) = (a/\theta_i : i \in I)$) if, and only if, $\cap_{i\in I}\theta_i = \Delta$.*

Definition 7. *An algebra is* subdirectly irreducible *(s.i., for short) if, for every representation $f : \mathbf{A} \to \Pi_{i\in I}\mathbf{A}_i$ as subdirect product, there exists j such that $\pi_j \circ f : \mathbf{A} \to \mathbf{A}_i$ is an isomorphism.*

Proposition 4. *$\mathbf{A}$ is s.i. iff $\cap(\mathrm{C}on(\mathbf{A}) - \{\Delta\}) \neq \Delta$ (in this case, $\cap(\mathrm{C}on(\mathbf{A}) - \{\Delta\})$ is called the* monolith*).*

Theorem 1. *(Birkhoff) Every algebra $\mathbf{A}$ is a subdirect product of s.i. algebras.*

An algebra $\mathbf{A}$ is *simple* if $\mathrm{C}on(\mathbf{A}) = \{\Delta, \nabla\}$. We have:

$$\text{Simple} \subseteq \text{Subdirectly Irreducible} \subseteq \text{Directly Indecomposable.}$$

Example 1. 1. Every two elements algebra is simple.
2. Every model of λ-calculus living in Scott continuous semantics is a simple combinatory algebra.
3. A vector space over a field is s.i. iff it is one-dimensional.
4. A Heyting algebra is s.i. iff there is a greatest element strictly below 1.
5. Every algebra of cardinality a prime number p is directly indecomposable.
6. The Stone representation theorem for Boolean algebras is a consequence of Theorem 1, because the algebra of truth values is the unique s.i. Boolean algebra.

Stone's representation theorem, perhaps the most distinctive result characterising Boolean algebras (or Boolean rings), can be generalised to a much larger class of algebras. The appropriate tool to attain this goal is the technique of Boolean products, which can be loosened to the notion of weak Boolean product to take care of somewhat less manageable cases. Pierce [22] proved that every commutative ring with unit is representable as a weak Boolean product of directly indecomposable rings. The technique of Boolean products underwent remarkable developments over the subsequent years (see e.g. [9, Ch. 4.8]), giving rise to further generalisations of Stone's theorem by Comer (covering the case of algebras with Boolean factor congruences [14]) and Vaggione [31].

Definition 8. *A* (weak) Boolean product *of a family $(\mathbf{A}_i)_{i\in I}$ of algebras is a subdirect product $\mathbf{A} \leq \prod_{i\in I}\mathbf{A}_i$, where I can be endowed with a Boolean space topology such that: (i) the set $\{i \in I : a_i = b_i\}$ is (open) clopen for all $a, b \in A$, and (ii) if $a, b \in A$ and $N \subseteq I$ is clopen, then the element c, defined by $c_i = a_i$ for $i \in N$ and $c_i = b_i$ for $i \in I - N$, belongs to A.*

3 Church Algebras

The key observation motivating the introduction of Church algebras [20] is that many algebras arising in completely different fields of mathematics, including Heyting algebras, rings with unit, combinatory algebras or λ-calculus, have a term operation q satisfying the fundamental properties of the if-then-else connective: $q(1,x,y) = x$ and $q(0,x,y) = y$. As simple as they may appear, these properties are enough to yield rather strong algebraic results, which will be applied to λ-calculus in the next section.

Definition 9. *We say that an algebra* $\mathbf{A}$ *is a* Church algebra *if it admits a ternary term operation* $q(x,y,z)$ *and two constants* $0, 1$ *such that the following identities are satisfied by* $\mathbf{A}$*:*

$$q(1,x,y) = x; \qquad q(0,x,y) = y.$$

A Church variety is a variety of Church algebras.

Example 2.

1. Rings with unit*:* $q(x,y,z) = xy + (1-x)z$*;*
2. Heyting algebras*:* $q(x,y,z) = (x \vee z) \wedge ((x \to 0) \vee y)$*;*
3. Combinatory algebras*:* $q(x,y,z) = (xy)z$, $1 = \mathbf{k}$ and $0 = \mathbf{sk}$*;*
4. Lambda calculus*:* $q(x,y,z) = (xy)z$*;* $1 = \lambda xy.x$ and $0 = \lambda xy.y$*.*

We denote by $\theta(a,b)$ the least congruence generated by the pair (a,b).

Lemma 3. *Let* $\mathbf{A}$ *be a Church algebra,* $(\phi, \overline{\phi})$ *be a pair of complementary factor congruences, and* e *be the unique element such that* $1\phi e\overline{\phi}0$*. Then, we have:*

(i) For every $a, b \in A$, $a\phi q(e,a,b)\overline{\phi}b$*.*
(ii) $\phi = \theta(1,e)$ *and* $\overline{\phi} = \theta(0,e)$*.*
(iii) The function $f_\phi(a,b) = q(e,a,b)$ *is a decomposition operator on* $\mathbf{A}$ *such that* $f_\phi(1,0) = e$*.*

In this definition we exploit an idea by Vaggione [31].

Definition 10. *An element* e *of a Church algebra* $\mathbf{A}$ *is* central *if* $\theta(1,e)$ *and* $\theta(0,e)$ *constitute a pair of complementary factor congruences of* $\mathbf{A}$*.* $\mathrm{Ce}(A)$ *denotes the set of central elements of the algebra* $\mathbf{A}$*.*

Proposition 5. *If* $\mathbf{A}$ *is a Church algebra and* $e \in A$*, then the following conditions are equivalent:*

1. *e is central;*
2. *For all* $a, b, \overline{a}, \overline{b} \in A$*:*

   $D1.$ $q(e,a,a) = a$
   $D2.$ $q(e, q(e,a,b), c) = q(e,a,c) = q(e,a,q(e,b,c))$
   $D3.$ $q(e, \sigma(\overline{a}), \sigma(\overline{b})) = \sigma(q(e,a_1,b_1), \dots, q(e,a_n,b_n))$ *(for every operation* σ*)*
   $D4.$ $q(e,1,0) = e.$

The partial ordering on $\mathrm{Ce}(A)$, defined by:

$$e \leq d \text{ if, and only if, } \theta(0,e) \subseteq \theta(0,d)$$

is a Boolean ordering. The meet, join and complementation operations are internally representable, and $0, 1$ are respectively the bottom and top element of this ordering.

Theorem 2. *Let* $\mathbf{A}$ *be a Church algebra.*

(i) The set $FC(\mathbf{A})$ *of factor congruences of* $\mathbf{A}$ *constitutes a Boolean sublattice of* $\mathrm{Con}(\mathbf{A})$.
(ii) The algebra $\mathrm{Ce}(\mathbf{A}) = (\mathrm{Ce}(A); \vee, \wedge, \neg, 0, 1)$*, where* $x \wedge y = q(x,y,0)$*,* $x \vee y = q(x,1,y)$ *and* $\neg x = q(x,0,1)$*, is a Boolean algebra isomorphic to the Boolean algebra of factor congruences of* $\mathbf{A}$.

3.1 The Stone Representation Theorem

A generic Church variety admits a weak Boolean product representation. The following theorem is a consequence of [9, Theorem 8.12].

Theorem 3. *Let* $\mathbf{A}$ *be a Church algebra,* S *be the Boolean space of maximal ideals of* $\mathrm{Ce}(\mathbf{A})$ *and* $f : A \to \Pi_{I \in S} A/\theta_I$ *be the map defined by*

$$f(a) = (a/\theta_I : I \in S),$$

where $\theta_I = \bigcup_{e \in I} \theta(0,e)$*. Then* f *gives a weak Boolean representation of* $\mathbf{A}$.

For the previous representation to be of some interest, we need to be in a position to provide additional information on its stalks. The following theorem is a consequence of [31, Theorem 8]. A new proof can be found in [27].

Theorem 4. *Let* $\mathcal{V}$ *be a Church variety. Then, the following conditions are equivalent:*

(i) For all $\mathbf{A} \in \mathcal{V}$*, the stalks* $\mathbf{A}/\theta_I$ *(*$I \in S$ *maximal ideal) are directly indecomposable.*
(ii) The class $\mathcal{V}_{DI}$ *of directly indecomposable members of* $\mathcal{V}$ *is a universal class.*

4 The Incompleteness Theorem of Lambda Calculus

The Stone representation theorem for combinatory algebras can be roughly summarised as follows: the directly indecomposable combinatory algebras are the "building blocks" in the variety of combinatory algebras. Then it is natural to investigate the class of models of λ-calculus, which are directly indecomposable as combinatory algebras (*indecomposable semantics*, for short).

In this section we show that the indecomposable semantics encompasses the Scott, stable and strongly stable semantics. In spite of this richness, we show that there exists a consistent λ-theory which is not the equational theory of an indecomposable model. The results in this section can be found in [21].

4.1 Scott Models are Simple Algebras

After Scott, several models of λ-calculus have been defined by order theoretic methods and classified into "semantics" according to the nature of their representable functions (see [3], for a survey on these semantics).

The *Scott-continuous semantics* corresponds to the class of λ-models having cpo's (complete partial orders) as underlying sets and representing all Scott continuous functions.

The *stable semantics* (Berry [5]) and the *strongly stable semantics* (Bucciarelli-Ehrhard [7]) are refinements of the Scott-continuous semantics which have been introduced to capture the notion of "sequential" continuous function. The underlying sets of the λ-models living in the stable (strongly stable) semantics are particular algebraic cpo's called *dI-domains* (*dI-domains with coherences*). These models represent all stable (strongly stable) functions between such domains. A function between dI-domains is stable if it is continuous and, furthermore, commutes with "infs of compatible elements". A strongly stable function between dI-domains with coherence, is a stable function preserving coherence.

In the next proposition we show that all models living in the main semantics are simple algebras.

Proposition 6. *All λ-models living in Scott, stable and strongly stable semantics are simple algebras.*

Proof. Let ϕ be a congruence on a Scott model $\mathbf{A}$ and let $a\phi b$ with $a \neq b$ and $a \not\leq b$. Since the continuous function g_c, defined by $g_c(x) =$ if $x \not\leq b$ then c else $\bot$, is representable in the model (for all c), we have: $c = g_c(a)\ \phi\ g_c(b) = \bot$, hence $c\phi\bot$ for all c. By the arbitrariness of c we get that ϕ is trivial.

Suppose now that $\mathbf{A}$ is a (strongly) stable model. Consider two elements $a, b \in A$ such that $a\phi b$, $a \neq b$ and $a \not\leq b$. There is a compact d such that $d \leq a$ and $d \not\leq b$. The step function $f_{d,c}$, defined by : $f_{d,c}(x) =$ if $d \leq x$ then c else $\bot$, is (strongly) stable for every element c. Then $c = f_{d,c}(a)\phi f_{d,c}(b) = \bot$.

4.2 Incompleteness

We now remark that the class of directly indecomposable combinatory algebras is a universal class.

To simplify the notation, in the following we write the combinators as λ-terms. For λ-terms t, u, we define the pair $[t, u] \equiv \lambda z.ztu$ and, for every sequence we define $[t_1, \ldots, t_n] \equiv [t_1, [t_2, \ldots, t_n]]$. Consider the following λ-terms:

- $P \equiv \lambda e.[\lambda x.exx, \lambda xyz.e(exy)z, \lambda xyz.exz, \lambda xyzu.e(xy)(zu), e(\lambda xy.x)(\lambda xy.y)]$;
- $Q \equiv \lambda e.[\lambda x.x, \lambda xyz.exz, \lambda xyz.ex(eyz), \lambda xyzu.exz(eyu), e]$.

We have that e is central in a combinatory algebra if, and only if, the equation $Pe = Qe$ holds.

Proposition 7. *The class of all directly indecomposable combinatory algebras is a universal class.*

Proof. The class of all directly indecomposable combinatory algebras is axiomatised by the following universal formula:

$$\forall e((Pe = Qe \to e = \lambda xy.x \vee e = \lambda xy.y) \wedge \neg(\lambda xy.x = \lambda xy.y)).$$

Corollary 1. *Let* **A** *be a combinatory algebra. Then* **A** *is isomorphic to a weak Boolean product of directly indecomposable combinatory algebras.*

Proof. By Proposition 7 and Theorem 4.

We are now ready to provide the algebraic incompleteness theorem. We recall that a class $\mathcal{C}$ of models of λ-calculus is *incomplete* if there exists a consistent λ-theory T such that $T \neq Th(\mathcal{M})$ for every $\mathcal{M} \in \mathcal{C}$.

Theorem 5. *The indecomposable semantics is incomplete.*

Proof. $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ can be consistently equated to every closed term. Let T_1 be the λ-theory generated by $\Omega = \lambda xy.x$ and T_2 be the λ-theory generated by $\Omega = \lambda xy.y$. Then Ω is central in the term model of $T_1 \cap T_2$.

Ω is central in a combinatory algebra **A** if, and only if, $\mathbf{A} \models P\Omega = Q\Omega$, where P and Q are the closed terms defined at the beginning of this section. Let $\mathcal{M}$ be a λ-model such that $Th(\mathcal{M}) = T_1 \cap T_2$. Then $\mathcal{M} \models P\Omega = Q\Omega$, because the identity $P\Omega = Q\Omega$ belongs to $T_1 \cap T_2$. Then Ω is a non-trivial central element of $\mathcal{M}$.

Corollary 2. *Scott, stable and strongly stable semantics are incomplete.*

5 The Order-Incompleteness Problem

The models of λ-calculus living in Scott, stable and strongly stable semantics are non-trivially ordered with a bottom element. However, it is also known that there are some models of the lambda calculus that cannot be non-trivially ordered (see [24,26,30]). In general, we define a combinatory algebra **A** to be *unorderable* if there does not exist a non-trivial partial order on A for which the application operation is monotone. Of course, an unorderable model can still arise from an order-theoretic construction, for instance as a subalgebra of some orderable model. The most interesting result has been obtained by Selinger [30], who, enough surprising, has shown the following result.

Theorem 6. *The term models of $\lambda\beta$ and $\lambda\beta\eta$ are unorderable.*

It follows that, if $\lambda\beta$ or $\lambda\beta\eta$ is the theory of a po-model, then the denotations of closed terms in that model are pairwise incomparable, i.e. the term denotations form an anti-chain.

Selinger's result can be used to show that the theory of a certain po-model $\mathcal{M}$ is not $\lambda\beta$ (or $\lambda\beta\eta$). It is sufficient to find out two closed λ-terms, whose denotations in the model $\mathcal{M}$ are related by the partial ordering. This technique has been successfully applied to some classes of models living in Scott continuous semantics. Bucciarelli-Salibra [8] have shown that $\lambda\beta$ cannot be the theory of a graph model, and Carraro-Salibra [10] have shown that $\lambda\beta\eta$ cannot be the theory of a reflexive Scott domain. Other results of this type can be found in Berline et al. [4].

The problem of unorderability led Selinger [30] to study the related question of absolute unorderability: a model is *absolutely unorderable* if it cannot be embedded in an orderable one. Plotkin conjectures in [24] that an absolutely unorderable combinatory algebra exists, but the question is still open whether this is so. Selinger has given in [30] a syntactic characterisation of the absolutely unorderable algebras in any variety of algebras in terms of the existence of a family of Mal'cev operators. Plotkin's conjecture is thus reduced to the question whether Mal'cev operators are consistent with the lambda calculus or combinatory logic.

Hereafter, we review Selinger's characterisation of absolutely unorderability.

Let $\mathbf{A}$ be an algebra of some variety $\mathcal{V}$. A preorder $\leq$ on $\mathbf{A}$ is *compatible* if it is monotone in each coordinate of every function symbol of $\mathcal{V}$. Then we have: (i) $\mathbf{A}$ is *unorderable* if it admits only equality as a compatible partial order; (ii) $\mathbf{A}$ is *absolutely unorderable* if, for every algebra $\mathbf{B} \in \mathcal{V}$ and every embedding $f : \mathbf{A} \to \mathbf{B}$, the algebra $\mathbf{B}$ is unorderable.

Let $\mathcal{V}$ be a variety, $\mathbf{A} \in \mathcal{V}$ and X be a set of indeterminates. We denote by $\mathbf{A}[X]$ the free extension of $\mathbf{A}$ in the variety $\mathcal{V}$. The algebra $\mathbf{A}[X]$ is defined up to isomorphism by the following universal mapping properties: (1) $A \cup X \subseteq A[X]$; (2) $\mathbf{A}[X] \in \mathcal{V}$; (3) for every $\mathbf{B} \in \mathcal{V}$, homomorphism $h : \mathbf{A} \to \mathbf{B}$ and every function $f : X \to B$, there exists a unique homomorphism $f : \mathbf{A}[X] \to \mathbf{B}$ extending h and f. When $X = \{x_1, \dots, x_n\}$ is finite, we write $\mathbf{A}[x_1, \dots, x_n]$ for $\mathbf{A}[X]$.

The following result by Selinger [30] characterises those algebras which are absolutely unorderable.

Theorem 7. *Let $\mathcal{V}$ be a variety. An algebra $\mathbf{A} \in \mathcal{V}$ is absolutely unorderable if, and only if, there exist a natural number $n \geq 1$ and ternary terms $p_1, \dots, p_n$ in the type of $\mathcal{V}$ such that the algebra $\mathbf{A}[x, y]$ satisfies the following identities, called (generalised) Mal'cev axioms:*

$$\begin{aligned} x &= p_1(x, y, y); \\ p_i(x, x, y) &= p_{i+1}(x, y, y) \quad (i = 1, \dots, n-1); \\ p_n(x, x, y) &= y. \end{aligned}$$

The following result was obtained by Plotkin-Simpson for $n = 1$ and by Plotkin-Selinger for $n = 2$ (see [30]).

Theorem 8. *For $n = 1$ and $n = 2$, the Mal'cev axioms are inconsistent with the lambda calculus.*

Proof. We prove the theorem for $n = 1$. Assume that $x = Fxyy$ and $Fxxy = y$ for a λ-term F. Let $Y \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ be the Curry fixpoint combinator. Then, for any λ-term M, define $\mu x.M \equiv Y(\lambda x.M)$. We write $\mu x_1 \dots x_n.M$

for $\mu x_1.(\mu x_2.(\cdots(\mu x_n.M)\cdots))$. Now let $A \equiv \mu yx.Fxyz$. Then we have $A = FAAz = z$ and $A = \mu x.FxAz = \mu x.Fxzz = \mu x.x = \Omega$, therefore $\Omega = z$.

The question of absolute unorderability can also be formulated in terms of theories, rather than models. In this form, Selinger [30] refers to it as the *order-incompleteness* question.

Definition 11. *A λ-theory is* order-incomplete *if it does not arise as the theory of a non-trivial po-model.*

The problem of order-incompleteness can be also characterised in terms of connected components of a partial ordering (minimal subsets which are both upward and downward closed): a λ-theory $\mathcal{T}$ is order-incomplete if, and only if, every po-model, having $\mathcal{T}$ as equational theory, is partitioned in an infinite number of connected components, each one containing exactly one element. In other words, the partial order is the equality.

Toward an answer to the order-incompleteness problem, the author has shown the following result in [26].

Theorem 9. *Every po-model $\mathcal{M}$ of the λ-theory T axiomatised by the equation $\Omega xx = \Omega$ is partitioned in an infinite number of connected components, each one containing at most the denotation of one λ-term (modulo T).*

The previous result has been improved in the forthcoming paper [11].

Corollary 3. *The semantics of λ-calculus given in terms of po-models with a finite number of connected components is incomplete.*

References

1. Barendregt, H.P.: The lambda calculus: Its syntax and semantics. North-Holland Publishing Co., Amsterdam (1984)
2. Bastonero, O., Gouy, X.: Strong stability and the incompleteness of stable models of λ-calculus. Annals of Pure and Applied Logic 100, 247–277 (1999)
3. Berline, C.: From computation to foundations via functions and application: The λ-calculus and its webbed models. Theoretical Computer Science 249, 81–161 (2000)
4. Berline, C., Manzonetto, G., Salibra, A.: Effective λ-models versus recursively enumerable λ-theories. Math. Struct. Comp. Science 19, 897–942 (2009)
5. Berry, G.: Stable Models of Typed Lambda-Calculi. In: Ausiello, G., Böhm, C. (eds.) ICALP 1978. LNCS, vol. 62, Springer, Heidelberg (1978)
6. Bigelow, D., Burris, S.: Boolean algebras of factor congruences. Acta Sci. Math. 54, 11–20 (1990)
7. Bucciarelli, A., Ehrhard, T.: Sequentiality and strong stability. In: LICS 1991. IEEE Computer Society Press (1991)
8. Bucciarelli, A., Salibra, A.: Graph lambda theories. Math. Struct. Math. Struct. in Comp. Science Science 18, 975–1004 (2008)
9. Burris, S., Sankappanavar, H.P.: A course in universal algebra. Springer, Berlin (1981)

10. Carraro, A., Salibra, A.: Reflexive Scott Domains are Not Complete for the Extensional Lambda Calculus. In: LICS 2009. IEEE Computer Society Press (2009)
11. Carraro, A., Salibra, A.: On the equational consistency of order-theoretic models of the λ-calculus (forthcoming 2012)
12. Church, A.: A set of postulates for the foundation of logic. Annals of Math. 2, 346–366 (1933)
13. Church, A.: The calculi of lambda conversion. Princeton University Press (1941)
14. Comer, S.: Representations by algebras of sections over Boolean spaces. Pacific Journal of Mathematics 38, 29–38 (1971)
15. Curry, H.B., Feys, R.: Combinatory Logic, vol. I. North-Holland Publishing Co. (1958)
16. Honsell, F., Ronchi della Rocca, S.: An approximation theorem for topological λ-models and the topological incompleteness of λ-calculus. Journal Computer and System Science 45, 49–75 (1992)
17. Kerth, R.: Isomorphisme et équivalence équationnelle entre modèles du λ-calcul. Thèse, Université de Paris 7 (1995)
18. Kerth, R.: On the construction of stable models of λ-calculus. Theoretical Computer Science 269, 23–46 (2001)
19. Lusin, S., Salibra, A.: The lattice of lambda theories. Journal of Logic and Computation 14, 373–394 (2004)
20. Manzonetto, G., Salibra, A.: From λ-Calculus to Universal Algebra and Back. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 479–490. Springer, Heidelberg (2008)
21. Manzonetto, G., Salibra, A.: Applying universal algebra to lambda calculus. Journal of Logic and Computation 20(4), 877–915 (2010)
22. Pierce, R.S.: Modules over Commutative Regular Rings. Memoirs of AMS (1967)
23. Pigozzi, D., Salibra, A.: Lambda abstraction algebras: representation theorems. Theoretical Computer Science 140, 5–52 (1995)
24. Plotkin, G.D.: On a question of H. Friedman. Information and Computation 126, 74–77 (1996)
25. Salibra, A.: On the algebraic models of lambda calculus. Theoretical Computer Science 249, 197–240 (2000)
26. Salibra, A.: Topological incompleteness and order incompleteness of the lambda calculus. ACM TOCL 4(3) (2003)
27. Salibra, A., Ledda, A., Paoli, F., Kowalski, T.: Boolean-like algebras. Algebra Universalis (to appear)
28. Scott, D.S.: Models for the λ-calculus. In: Toposes, Algebraic Geometry and Logic. LNM, vol. 274, Springer (1972)
29. Scott, D.S.: Lambda calculus: some models, some philosophy. In: The Kleene Symposium. North-Holland (1980)
30. Selinger, P.: Order-incompleteness and finite lambda reduction models. Theoretical Computer Science 309, 43–63 (2003)
31. Vaggione, D.: Varieties in which the Pierce stalks are directly indecomposable. Journal of Algebra 184, 424–434 (1996)

A Toolkit for Proving Limitations of the Expressive Power of Logics

Nicole Schweikardt

Goethe-University Frankfurt, Frankfurt am Main, Germany
schweika@cs.uni-frankfurt.de
http://www.tks.cs.uni-frankfurt.de/schweika

Zero-one laws, Ehrenfeucht-Fraïssé games, locality results, and logical reductions belong to the, by now, standard methods of Finite Model Theory, used for showing non-expressibility in certain logics (cf., e.g., the textbooks [1,2] or the entries in the Encyclopedia of Database Systems [3]).

More recently, the close connections between logic and circuits, along with strong lower bound results obtained in circuit complexity, have led to new lower bounds on the expressiveness of logics (cf., e.g., [4,5,6,7]). In particular, [4] solved a long standing open question of Finite Finite Model Theory, asking about the strictness of the bounded variable hierarchy of first-order logic on finite ordered graphs.

To characterise the precise expressive power of logics on particularly well-behaved classes of finite structures, the following "algebraic" approach has recently been very successful (cf., e.g., [8,9,10]): The first step is to identify a number of closure properties that the classes of structures defined by sentences of the logic exhibit. The second step is to identify another logic that is characterised by these closure properties. An example of this methodology is a result of [8], stating that on successor-based strings, plain first-order logic (FO, for short) is as expressive as order-invariant FO. The proof of [8] proceeds by showing that languages definable in order-invariant FO are aperiodic and closed under swaps. Then, Beauquier and Pin's characterisation [11] of FO by aperiodicity and closure under swaps immediately leads to the desired result.

The aim of this talk is to give an overview of the above mentioned methods for proving limitations of the expressive power of logics.

References

1. Libkin, L.: Elements of Finite Model Theory. Springer (2004)
2. Ebbinghaus, H.-D., Flum, J.: Finite model theory. Springer (1995)
3. Liu, L., Özsu, M.T. (eds.): Encyclopedia of Database Systems. Springer (2009)
4. Rossman, B.: On the constant-depth complexity of k-clique. In: Proc. 40th Annual ACM Symposium on the Theory of Computing (STOC 2008), pp. 721–730. ACM (2008)
5. Anderson, M., van Melkebeek, D., Schweikardt, N., Segoufin, L.: Locality of Queries Definable in Invariant First-Order Logic with Arbitrary Built-in Predicates. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 368–379. Springer, Heidelberg (2011)

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 46–47, 2012.

6. Anderson, M., van Melkebeek, D., Schweikardt, N., Segoufin, L.: Locality from circuit lower bounds. To appear in SIAM Journal on Computing (2012)
7. Kikot, S., Kontchakov, R., Podolskii, V., Zakharyaschev, M.: Exponential Lower Bounds and Separation for Query Rewriting. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 263–274. Springer, Heidelberg (2012)
8. Benedikt, M., Segoufin, L.: Towards a characterization of order-invariant queries over tame structures. Journal of Symbolic Logic 74(1), 168–186 (2009)
9. Schweikardt, N., Segoufin, L.: Addition-invariant FO and regularity. In: Proc. 25th IEEE Symposium on Logic in Computer Science (LICS), pp. 273–282 (2010)
10. Harwath, F., Schweikardt, N.: Regular tree languages, cardinality predicates, and addition-invariant FO. In: Proc. 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012). LIPIcs, vol. 14, pp. 489–500 (2012)
11. Beauquier, D., Pin, J.-E.: Factors of words. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 63–79. Springer, Heidelberg (1989)

How to Reconstruct a Genome

Esko Ukkonen*

Department of Computer Science, University of Helsinki, Finland
Esko.Ukkonen@cs.helsinki.fi

Abstract. Since its early formulations (e.g., [1]), the genome assembly problem has attracted lots of interest from algorithm theoretic as well as from algorithm engineering point of view. In this problem, which is an inversion problem by nature, one is asked to reconstruct the entire DNA sequence from the short, randomly picked sequence fragments that a DNA sequencing instrument is able to read [2]. With the invent of current high-throughput sequencers producing such fragment reads in massive amounts, there is in molecular biology research a pronounced call for an accurate and fast reconstruction procedure.

It is customary to structure a reconstruction procedure into the following major steps: (1) Error correction of the fragments; (2) Finding pairwise overlaps between the fragments and representing the overlaps as a graph; (3) Constructing approximate superstrings, called *contigs*, for the fragments; (4) Constructing a linear order, called a *scaffold*, of the contigs. All steps are algorithmically challenging. Noisy data and intricate repetition structure of the target genome cause added difficulties.

The talk attempts to give an overall picture of the genome assembly process and its algorithmic aspects emphasizing some recent developments in error correction [3], contig assembly, and scaffolding [4]. We also try to convey experiences from a major undertaking of *de novo* sequencing of a higher organism, Glanville fritillary butterfly *Melitea cinxia*. (A collaboration with I. Hanski, www.helsinki.fi/science/metapop/index.htm).

References

1. Peltola, H., Söderlund, H., Tarhio, J., Ukkonen, E.: Algorithms for Some String Matching Problems Arising in Molecular Genetics. In: IFIP Congress, pp. 59–64 (1983)
2. Myers, G.: Whole-Genome DNA Sequencing. IEEE Computing in Science and Engineering 1, 33–43 (1999)
3. Salmela, L., Schröder, J.: Correcting errors in short reads by multiple alignments. Bioinformatics 27, 1455–1461 (2011)
4. Salmela, L., Mäkinen, V., Välimäki, N., Ylinen, J., Ukkonen, E.: Fast scaffolding with small independent mixed integer programs. Bioinformatics 27, 3259–3265 (2011)

* Supported by the Academy of Finland, grant 7523004 (Algorithmic Data Analysis).

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, p. 48, 2012.

Simple Models for Recursive Schemes

Igor Walukiewicz*

LaBRI, CNRS/Universit Bordeaux, France

Abstract. Higher-order recursive schemes are abstract forms of programs where the meaning of built-in constructs is not specified. The semantics of a scheme is an infinite tree labeled with built-in constructs. The research on recursive schemes spans over more than forty years. Still, central problems like the equality problem, and more recently, the model checking problem for schemes remain very intriguing. Even though recursive schemes were originally though of as a syntactic simplification of a fragment of the lambda calculus, we propose to go back to lambda calculus to study schemes. In particular, for the model checking problem we propose to use standard finitary models for the simply-typed lambda calculus.

1 Introduction

A recursive scheme is a set of equations over a fixed functional signature. On the left of an equation we have a function symbol and on the right a term that is its intended meaning. Consider the following examples borrowed from [10]:

$$\begin{aligned}\mathbf{F}(x) &\equiv \textbf{if } x=0 \textbf{ then } 1 \textbf{ else } \mathbf{F}(x-1)\cdot x,\\ \mathbf{F}(x) &\equiv C\big((Zx),\ A,\ M(\mathbf{F}(P(x)),x)\big).\end{aligned}$$

The first is the usual recursive definition of factorial. The second is the same definition in an abstract form where abstract function names have been used instead of the ones with a well established meaning. Observe that the reverse function

$$\mathit{Rev}(x) \equiv \textbf{if } x=\text{nil} \textbf{ then } \text{nil} \textbf{ else } \textbf{append}(\mathit{Rev}(\text{tl}(x)), \mathit{hd}(x))$$

has the same abstract form as factorial: the pattern of function calls is the same in the two cases.

The above schemes are of order 1 as functions they define work on elements of a basic type. An order 2 scheme allows to express for example *map* function that applies a given function f to every element of the list l:

$$\mathit{map}(f,x) \ \equiv\ \textbf{if } l=\mathit{nil} \textbf{ then } \mathit{nil} \textbf{ else } \textbf{cons}(f(\textbf{head}(l)), \mathit{map}(f, \textbf{tail}(l)))$$

This is a scheme of order 2 because its argument f is a function on elements of a base type. Higher-order recursive schemes are schemes of arbitrary finite order.

* Supported by ANR 2010 BLAN 0202 01 FREC.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 49–60, 2012.

Recursive schemes are about abstract forms of programs where the meaning of constants is not specified. In consequence, the meaning of a scheme is a potentially infinite tree labelled with constants obtained from the unfolding of the recursive definition. Let us immediately note that we impose a typing disciple on terms. In this form recursive schemes are essentially a different presentation of simply typed lambda calculus with fixpoint operators. Indeed, recursive definitions as in the examples above can be reformulated using the fix point operator explicitly.

Recursion schemes were originally proposed by Ianov as a canonical programming calculus for studying program transformation and control structures [18]. The study of recursion on higher types as a control structure for programming languages was started by Milner [30] and Plotkin [34]. Program schemes for higher-order recursion were introduced by Indermark [19]. Higher-order features allow for compact high-level programs. They have been present since the beginning of programing, and appear in modern programming languages like C++, Haskell, Javascript, Python, or Scala. Higher-order features allow to write code that is closer to specification, and in consequence to obtain a more reliable code. This is particularly useful in the context when high assurance should come together with very complex functionality. Telephone switches, simulators, translators, statistical programs operating on terabytes of data, have been successfully implemented using functional languages[1].

Research on higher-order schemes has spanned many decades. Recursive schemes appear as an intermediate step in semantics of programming languages. Indeed to compute the semantics of a program one can first compute the infinite tree of behaviors it generates, and then apply an interpretation operation giving the meaning of constants [38,31]. This view brought to the scene the equality problem for schemes [27,11,9,39], whose decidability in the higher-order case is still open. It has then been discovered that schemes have many links with language theory, in particular with context-free and context-sensitive languages [16,9,13]. More recently, it has been understood that recursive schemes give important classes of trees with decidable MSOL theory which makes them interesting from the point of view of automatic verification [21,32].

Let us see an example illustrating why recursive schemes are valuable abstractions of programs. Simple while loops with the usual arithmetic operations are Turing complete. This of course does not make all other programming constructs obsolete. Yet, in the light of this universality result we need to look for some other convincing framework where we could show that concepts like recursive procedures or higher-order types bring something new. The idea is to abstract from the meaning of build-in operations, or in other words to consider them as uninterpreted function symbols. This way a program is stripped to its control structure: it becomes a program scheme. Coming back to our example with while programs, once influence of the arithmetic is stripped away, one can

[1] For some examples see "Functional programming in the real world" `http://homepages. inf.ed.ac.uk/wadler/realworld/`

formally show that recursive procedures are indeed more powerful than a simple while loop.

The above example shows that recursive schemes are an insightful intermediate step in giving a denotational semantics of a program. This makes the study of equality or verification problems for schemes interesting. Especially in view that these problems may be decidable for schemes while they are not for interpreted programs.

Equality Problem. Given two schemes decide if they generate the same tree.
Model-Checking Problem. Given a schema and a formula of monadic second-order logic decide if the formula holds in the tree generated by the scheme.

Equality of two schemes gives a provably correct program transformation rule. Equality of first-order schemes is equivalent to equality of deterministic pushdown automata [9]. Hence it is decidable by the result of Sénizergues [39]. Some properties of a program can be expressed in terms of the tree generated by the associated scheme. For example, resource usage patterns can be formulated in fragments of monadic second-order logic and verified over such trees [23]. This is possible thanks to the fact that MSOL model checking is decidable for trees generated by higher-order recursive schemes [32].

The meaning of a recursive scheme is an infinite ranked tree. A natural question is then what are these trees. Are there other characterizations of these objects, and what are their properties? First answers came from language theory. Damm [13] has shown that considered as word generating devices, a class of schemes called safe is equi-expressive with higher-order indexed languages introduced by Aho and Maslov [3,28]. Those languages in turn have been know to be equivalent to higher-order pushdown automata of Maslov [29]. Later it has been shown that trees generated by higher-order safe schemes are the same as those generated by higher-order pushdown automata [21]. This gave rise to so called Caucal hierarchy [8] and its numerous characterizations [7]. The safety restriction has been tackled much more recently. First, because it has been somehow implicit in a work of Damm [13], and only brought on the front stage by Knapik, Niwiński, and Urzyczyn [21]. Secondly, because it required new insights in the nature of higher-order computation. Pushdown automata have been extended with so called panic operation [22,2]. This permitted to characterize trees generated by schemes of order two. Later this operation has been extended to all higher order stacks, and called collapse. Higher-order stack automata with collapse characterise recursive schemes at all levels [17]. The fundamental question whether collapse operation adds expressive power has been answered affirmatively only very recently by Parys: there is a tree generated by an order 2 scheme that cannot be generated by a higher-order stack automaton without collapse [33].

In this paper we will concentrate on the model checking problem. There has been a steady progress on this problem, to the point that there are now tools implementing verification algorithms for higher-order programs [24]. In contrast, the most classical problem, namely the equivalence problem, remains as a great challenge. The fundamental result of Sénizergues [39] and subsequent revisits of the proof by Stirling [41], Sénizergues [40], and Jancar [20] give an algorithm to test equivalence of schemes of order 1. Yet this algorithm is not only non-elementary but also gives a strong impression that its average performance would be close to non-elementary too. In contrast, while also non-elementary in the worst case, the algorithms for model checking are capable of solving nontrivial verification problems. Let us also recall that we know that the complexity of the model checking problem is non-elementary [15,6,25], while no non-trivial lower bounds are known for the equivalence problem.

The objective of this short paper is to propose an approach to the model checking problem for schemes. The idea is to work with a richer syntax of simply typed lambda calculus with fixpoints, and to construct suitable finitary models that give the answer to the problem. We will carry out this program only for a relatively small subclass of all monadic-second order properties: the same as considered by Aehlig [1]. The next section introduces necessary notions, in particular that of a Böhm tree of a term. In the following section we show how, given an automaton expressing the property, to construct a desired finitary model from which we can read properties of Böhm trees of terms.

2 Simply Typed Lambda Calculus and Recursive Schemes

Instead of introducing higher-order recursive schemes directly we prefer to start with simply typed lambda calculus with fixpoints, *λY-calculus*. The two formalisms are essentially equivalent for the needs of this paper, but we will prefer work with the later one. It gives us an explicit notion of reduction, and brings the classical notion of Böhm tree [4] that can be used directly to define the meaning of a scheme.

The *set of types* $\mathcal{T}$ is constructed from a unique *basic type* 0 using a binary operation $\to$. Thus 0 is a type and if α, β are types, so is $(\alpha \to \beta)$. The order of a type is defined by: $order(0) = 1$, and $order(\alpha \to \beta) = max(1 + order(\alpha), order(\beta))$.

A *signature*, denoted Σ, is a set of typed constants, that is symbols with associated types from $\mathcal{T}$. We will assume that for every type $\alpha \in \mathcal{T}$ there are constants ω^α and $Y^{(\alpha\to\alpha)\to\alpha}$. A constant $Y^{(\alpha\to\alpha)\to\alpha}$ will stand for a fixpoint operator, and ω^α for undefined. Of special interest to us will be *tree signatures* where all constants other than Y and ω have order at most 2. Observe that types of order 2 have the form $0^i \to 0$ for some i; the later is a short notation for $0 \to 0 \to \cdots \to 0 \to 0$, where there are $i + 1$ occurrences of 0.

The set of *simply typed λ-terms* is defined inductively as follows. A constant of type α is a term of type α. For each type α there is a countable set of variables $x^\alpha, y^\alpha, \dots$ that are also terms of type α. If M is a term of type β and x^α a variable of type α then $\lambda x^\alpha.M^\beta$ is a term of type $\alpha \to \beta$. Finally, if M is of type $\alpha \to \beta$ and N is a term of type α then MN is a term of type β.

The usual operational semantics of the λ-calculus is given by β-reduction. To give the meaning to fixpoint constants we use δ-reduction ($\to_\delta$).

$$(\lambda x.M)N \to_\beta M[N/x] \qquad YM \to_\delta M(YM).$$

We write $\to^*_{\beta\delta}$ for the reflexive and transitive closure of the sum of the two relations. This relation defines an operational equality on terms. We write $=_{\beta\delta}$ for the smallest equivalence relation containing $\to^*_{\beta\delta}$. It is called *$\beta\delta$-equality.*

Thus, the operational semantics of the λY-calculus is the $\beta\delta$-reduction. It is well-known that this semantics is confluent and enjoys subject reduction (*i.e.* the type of terms is invariant under computation). So every term has at most one normal form, but due to δ-reduction there are terms without a normal form. It is classical in the lambda calculus to consider a kind of infinite normal form that by itself is an infinite tree, and in consequence it is not a term of λY [4,14,5]. We define it below.

A *Böhm tree* is an unranked ordered, and potentially infinite tree with nodes labelled by ω^α or terms of the form $\lambda x_1. \ldots x_n.N$; where N is a variable or a constant, and the sequence of lambda abstractions is optional. So for example x^0, $\lambda x.w^0$ are labels, but $\lambda y^0.x^{0\to 0}.\ y^0$ is not.

Definition 1. *A* Böhm tree *of a term M is obtained in the following way.*

- *If $M \to^*_{\beta\delta} \lambda \boldsymbol{x}.N_0N_1 \ldots N_k$ with N_0 a variable or a constant then $BT(M)$ is a tree having root labelled by $\lambda\boldsymbol{x}.N_0$ and having $BT(N_1), \ldots, BT(N_k)$ as subtrees.*
- *Otherwise $BT(M) = \omega^\alpha$, where α is the type of M.*

Observe that a term M has a $\beta\delta$-normal form if and only if $BT(M)$ is a finite tree without ω constants. In this case the Böhm tree is just another representation of the normal form. Unlike in the standard theory of the λ-calculus we will be rather interested in terms with infinite Böhm trees.

Recall that in a tree signature all constants except of Y and ω are of order at most 2. A closed term without λ-abstraction and Y over such a signature is just a finite tree, where constants of type 0 are in leaves and constants of a type $0^k \to 0$ are labels of inner nodes with k children. The same holds for Böhm trees:

Lemma 1. *If M is a closed term of type* 0 *over a tree signature then $BT(M)$ is a potentially infinite tree whose leaves are labeled with constants of type* 0 *and whose internal nodes with k children are labelled with constants of type* $0^k \to 0$.

Higher-order recursive schemes use somehow simpler syntax: the fixpoint operators are implicit and so is the lambda-abstraction. A recursive scheme over a finite set of nonterminals $\mathcal{N}$ is a collection of equations, one for each nonterminal. A nonterminal is a typed functional symbol. On the left side of an equation we have a nonterminal, and on the right side a term that is its meaning. For a formal definition we will need the notion of an *applicative term*, that is a term constructed from variables and constants, other than Y and ω, using the application operation. Let us fix a tree signature Σ, and a finite set of typed nonterminals $\mathcal{N}$. A higher-order recursive scheme is a function $\mathcal{R}$ assigning to every nonterminal $F \in \mathcal{N}$, a term $\lambda \boldsymbol{x}.M_F$ where: (i) M_F is an applicative term, (ii) the type of $\lambda \boldsymbol{x}.M_F$ is the same as the type of F, and (iii) the free variables of M are among $\boldsymbol{x}$ and $\mathcal{N}$. For example, the following is a scheme of the map function:

$$map^{(0\to 0)\to 0\to 0} \equiv !\lambda f^{0\to 0}.\lambda l^0.\,\mathbf{if}(l = nil,\ nil,\ \mathbf{cons}(f(\mathbf{head}(l)), map(f, \mathbf{tail}(l))))$$

The translation from a recursive scheme to a lambda-term is given by a standard variable elimination procedure, using the fixpoint operator Y. Suppose $\mathcal{R}$ is a recursive scheme over a set of nonterminals $\mathcal{N} = \{F_1, \dots, F_n\}$. The term T_n representing the meaning of the nonterminal F_n is obtained as follows:

$$\begin{aligned} T_1 =&Y(\lambda F_1.\mathcal{R}(F_1)) \\ T_2 =&Y(\lambda F_2.\mathcal{R}(F_2)[T_1/F_1]) \\ &\vdots \\ T_n =&Y(\lambda F_n.(\dots((\mathcal{R}(F_n)[T_1/F_1])[T_2/F_2])\dots)[T_{n-1}/F_{n-1}]) \end{aligned} \tag{1}$$

The translation (1) applied to the recursion scheme for *map* gives a term:

$$\begin{aligned} &Y(\lambda map^{(0\to 0)\to 0\to 0}.\lambda f^{0\to 0}.\lambda l^0. \\ &\qquad\qquad \mathbf{if}\ (l = nil)\ nil\ \big(\mathbf{cons}\ (f(\mathbf{head}(l)))\ (map\ f\ (\mathbf{tail}(l)))\big) \end{aligned}$$

This time we have used λ-calculus way of parenthesising expressions.

We will not recall here a rather lengthy definition of a tree generated by a recursive scheme referring the reader to [21,13]. For us it will be sufficient to say that it is the Böhm tree of a term obtained from the above translation. For completeness we state the following.

Lemma 2. *Let $\mathcal{R}$ be a recursion scheme and let F_n be one of its nonterminals. A term T_n obtained by the translation* (1) *is such that $BT(T_n)$ is the tree generated by the scheme from nonterminal F_n.*

3 Models for Model Checking

As we have said in the introduction, in order to study the model checking problem it is very helpful to understand better what are trees generated by recursive schemes. The proof of Ong of decidability of the model checking [32] relies on game semantics to understand the structure of such trees. Later proofs rely on higher-order pushdown automata with panic [22,2], for order 2, and for collapse for all orders [17]. Krivine machines give also a good representation of trees, and another proof of Ong's result [37]. In all these approaches to model checking, one first obtains a characterisation of trees generated by recursive schemes and then solves the model checking problem using this characterisation.

There is another, more denotational, way of approaching the model checking problem. One can analyse the scheme from the point of view of a given property without constructing the generated tree first. This approach can be carried out with a help of a rich typing discipline. For a given property one constructs a set of types and typing rules such that the scheme is typable if and only if the tree it generates satisfies the property. This approach has been successfully carried out for properties expressed by automata with a trivial acceptance condition [23]. The extension to all parity automata required much more work and complicated substantially the approach [26]. It is worth mentioning that the discovery that model checking with respect to automata with trivial conditions is much easier than the general case is due to Aehlig [1]. His proof uses a mixture of semantics and typing systems. The approach used by Kobayashi [23] is based on intersection types refining simple types that allows to capture regular properties of terms. This kind of technique has been probably initiated by Salvati [35].

Instead of typings we propose to use models of simply-typed lambda calculus. For simply-typed lambda calculus without fixpoints the intersection types and standard models are in some sense dual [36]. Our construction hints that this can be also the case in the presence of fixpoint operators. Moreover the model based approach has a striking simplicity in the case we present here.

Since the translation from recursive schemes to the λY-calculus is very direct, it is straightforward to interpret a recursive scheme in a model of the λY-calculus. Once this step is taken, it is even more natural to work directly with the lambda calculus. So starting with a property we would like construct a model such that the value of a term in the model determines if the Böhm tree of the term satisfies the property. This is formalised in Theorem 1 below.

Let us consider finitary models of λY-calculus. We concentrate on those where Y is interpreted as the greatest fixpoint.

Definition 2. *A* GFP-model *of a signature Σ is collection of finite complete lattices, one for each type, together with a valuation of constants: $\mathcal{D} = \langle \{D^\alpha\}_{\alpha\in\mathcal{T}}, \rho\rangle$. The model is required to satisfy the following conditions:*

- *D^0 is a finite lattice;*
- *for every type $\alpha \to \beta \in \mathcal{T}$, $D^{\alpha\to\beta}$ is the lattice of monotone functions from D^α to D^β ordered coordinatewise;*
- *If $c \in \Sigma$ is a constant of type α then $\rho(c)$ is an element of D^α. For every $\alpha \in \mathcal{T}$ it must be that case that $\rho(\omega^\alpha)$ is the greatest element of D^α. Moreover, $\rho(Y^{(\alpha\to\alpha)\to\alpha})$ should be a function assigning to every function $f \in D^{\alpha\to\alpha}$ its greatest fixpoint.*

Observe that every D^α is finite, hence all the greatest fixpoints exists without any additional assumptions on the lattice.

A *variable assignment* is a function υ associating to a variable of type α an element of D^α. If d is an element of D^α and x^α is a variable of type α then $\upsilon[d/x^\alpha]$ denotes the valuation that assigns d to x^α and that is identical to υ otherwise.

The *interpretation of a term M* of type α in the model $\mathcal{D}$ under valuation υ is an element of D^α denoted $[\![M]\!]^\upsilon_{\mathcal{D}}$. The meaning is defined inductively:

- $[\![c]\!]^{\upsilon}_{\mathcal{D}} = \rho(c)$
- $[\![x^\alpha]\!]^{\upsilon}_{\mathcal{D}} = \upsilon(x^\alpha)$
- $[\![MN]\!]^{\upsilon}_{\mathcal{D}} = [\![M]\!]^{\upsilon}_{\mathcal{D}}[\![N]\!]^{\upsilon}_{\mathcal{D}}$
- $[\![\lambda x^\alpha.M]\!]^{\upsilon}_{\mathcal{D}}$ is a function mapping an element $d \in D^\alpha$ to $[\![M]\!]^{\upsilon[d/x^\alpha]}_{\mathcal{D}}$.

As usual, we will omit subscripts or superscripts in the notation of the semantic function if they are clear from the context.

Of course a GFP model is sound with respect to $\beta\delta$-equality. Hence two equal terms have the same semantics in the model. For us it is important that a stronger property holds: if two terms have the same Böhm trees then they have the same semantics in the model.

Lemma 3. *For every GFP-model $\mathcal{D}$, and closed terms M, N of λY-calculus: if $BT(M) = BT(N)$ then $[\![M]\!]_{\mathcal{D}} = [\![N]\!]_{\mathcal{D}}$.*

Before stating the theorem we need to explain how properties of Böhm trees are specified. We will consider tree signatures, so Böhm trees of closed terms of type 0 are just ranked, potentially infinite trees (cf. Lemma 1). To express properties of such trees we can use monadic second-order logic, or an equivalent formalism of finite automata on infinite trees. We will use the later since it allows for an easy formulation of the important restriction we will make. We will consider only automata with trivial acceptance condition. This means that every run of such an automaton is accepting. So the trees that are not accepted are those over which the automaton does not have a run. We define this concept below.

Let us fix a tree signature Σ. Recall that this means that apart from ω and Y all constants have order at most 2. For simplicity of notation we will assume that constants of order 2 have type $0 \to 0 \to 0$. In this case, by Lemma 1, Böhm trees are potentially infinite binary trees. Let Σ_1 be the set of constants of order 1, hence of type 0, and Σ_2 the set of constants of order 2, hence of type $0 \to 0 \to 0$.

Definition 3. *A* finite automaton with trivial acceptance condition *over the signature $\Sigma = \Sigma_1 \cup \Sigma_2$ is*

$$\mathcal{A} = \langle Q, \Sigma, q^0 \in Q, \delta_1 : Q \times \Sigma_1 \to \{\mathit{ff}, \mathit{tt}\}, \delta_2 : Q \times \Sigma_2 \to \mathcal{P}(Q^2)\rangle$$

where Q is a finite set of states and $q^0 \in Q$ is the initial state.

Observe that the type of δ_1 logically follows from the fact that a constant of type 0 is a "function with no arguments", hence the range of δ_1 should be $\mathcal{P}(Q^0)$ that is more intuitively presented as $\{\mathit{ff}, \mathit{tt}\}$.

Automata will run on Σ-labelled binary trees that are partial functions $t : \{0,1\}^* \to \Sigma$ such that their domain is a binary tree, and $t(u) \in \Sigma_1$ if u is a leaf, and $t(u) \in \Sigma_2$ otherwise.

A *run of $\mathcal{A}$ on t* is a partial mapping $r : \{0,1\}^* \to Q$ with the same domain as t an such that:

- $r(\varepsilon) = q^0$, here ε is the root of t.
- $(r(u0), r(u1)) \in \delta_2(t(u), r(u))$ if u is an internal node.

A run is *accepting* if for every leaf u of t either $\delta_1(r(u), t(u)) = tt$, or $t(u) = \omega^0$. The later condition means that the automaton accepts in leaves labelled by ω^0 constant. A tree is *accepted by* $\mathcal{A}$ if there is an accepting run on the tree. The *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of trees accepted by $\mathcal{A}$.

Observe that our automata have acceptance conditions on leaves, expressed with δ_1, but do not have acceptance conditions on infinite paths. The clause about always accepting in leaves labelled ω^0 has some consequences. In particular, with these automata we cannot define a set of terms that do not have ω^0 in their Böhm tree. This restriction appears also in the works of Aehlig [1] and Kobayashi [23].

Theorem 1. *For every automaton with trivial acceptance conditions $\mathcal{A}$ there is a GFP-model $\mathcal{D}_\mathcal{A}$ and a set $F_\mathcal{A} \subseteq D^0_\mathcal{A}$ such that for every closed term M of type* 0*:*

$$BT(M) \in L(\mathcal{A}) \quad \textit{iff} \quad [\![M]\!]_{\mathcal{D}_\mathcal{A}} \in F_\mathcal{A}.$$

For $\mathcal{D}_\mathcal{A}$ we take a GFP model with the domain for the base type being the set of subsets of the set of states of $\mathcal{A}$: $D^0_\mathcal{A} = \mathcal{P}(Q)$. This choice determines all D^α. It remains to define interpretation of constants other than ω or Y. A constant $c \in \Sigma$ of type 0 is interpreted as the set $\{q : \delta_1(q, c) = tt\}$. A constant $a \in \Sigma$ of type $0 \to 0 \to 0$ is interpreted as the function whose value on $(S_0, S_1) \in \mathcal{P}(Q)^2$ is $\{q : \delta_2(q, a) \in S_0 \times S_1\}$. Finally, we put $F_\mathcal{A} = \{S : q^0 \in S\}$; recall that q^0 is the initial state of $\mathcal{A}$. The proof of the theorem is rather easy. Lemma 3 allows to work with Böhm trees instead of terms. Then one can use approximations of an infinite tree by its finite prefixes.

Corollary 1. *Given a closed λY-term M of type* 0 *and an automaton with a trivial acceptance condition $\mathcal{A}$ it is decidable if $BT(M)$ is accepted by $\mathcal{A}$.*

The decidability follows immediately from Theorem 1, and the fact that $\mathcal{D}_\mathcal{A}$ is a finitary model, so the semantics of a term can be computed just using the definition. The answer is positive if an only if the obtained meaning is in $F_\mathcal{A}$. Of course computing the meaning of a term may be computationally difficult. The sizes of domains grow fast with the order of the type: the size of $D^{\alpha\to\beta}_\mathcal{A}$ is exponentially bigger than the size of $D^\alpha_\mathcal{A}$. It is known that the model checking problem is nonelementary [15,6,25], and at closer inspection the worst case complexity of the approach presented here is on a par with other approaches.

It is not clear how to extend this method to automata with parity conditions. What is straightforward to do is to extend it to automata that are dual to automata with trivial acceptance conditions. Automata with trivial acceptance conditions are in fact equivalent in expressive power to alternating parity automata whose all states have rank 0. The dual automata are alternating automata whose all states have rank 1. Let us call them *rank* 1 *automata.* In particular rank 1 automata accept the complements of the languages accepted by automata with trivial conditions.

In the theorem above, we have used GFP models: we have interpreted Y constants as the greatest fixpoint operators. To capture the power of rank 1 automata we take LFP models where Y's are interpreted as the least fixpoint operators, and ω's as the least elements in corresponding lattices. Dualizing the argument we get the corresponding result

Theorem 2. *For every rank* 1 *automaton* $\mathcal{A}$ *there is a LFP model* $\mathcal{E}_\mathcal{A}$ *and a set* $F_\mathcal{A} \subseteq E^0$ *such that for every closed term* M *of type* 0*:*

$$BT(M) \in L(\mathcal{A}) \quad \textit{iff} \quad [\![M]\!]_{\mathcal{E}_\mathcal{A}} \in F_\mathcal{A}.$$

4 Conclusions

We have argued that recursive schemes are a fundamental notion that has appeared in a number of areas of computer science like: schematology [27], language theory [13], and verification [21,32]. For this reason there is a number of different approaches to view and study schemes. We have suggested yet another one in this paper. The mixture of the lambda calculus, language theory, and semantics makes this subject a promising playground for all three disciplines.

Let us hint yet another, more logic based, approach to understand schemes. Instead of trying to find a new device capable of generating the same trees as schemes do, one can look for operations on trees or graphs. In the case of safe schemes this program has been successfully carried out resulting in what is now called Caucal hierarchy [8]. This is the set of trees obtained from the one node tree by operations of unfolding of a graph into a tree [12], and MSOL interpretations. Since both these operations preserve MSOL decidability, we immediately obtain that all the trees in the Caucal hierarchy have decidable MSOL theory. It can be then shown that these are up to simple MSOL interpretations precisely the trees generated by recursive schemes. A number of other characterisations of this class of trees exist: via rewriting rules, using Muchnik's unfolding operation, using higher order pushdown automata (without collapse) [7]. By the result of Parys [33] we know that there are trees generated by recursive schemes that do not belong to Caucal hierarchy. One can imagine that there exists some operation preserving decidability of MSOL theories that allows to obtain all tress generated by higher-order recursive schemes in the same way as the unfolding operation does for safe schemes.

References

1. Aehlig, K.: A finite semantics of simply-typed lambda terms for infinite runs of automata. Logical Methods in Computer Science 3(1), 1–23 (2007)
2. Aehlig, K., de Miranda, J.G., Ong, C.-H.L.: The Monadic Second Order Theory of Trees Given by Arbitrary Level-Two Recursion Schemes Is Decidable. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 39–54. Springer, Heidelberg (2005)
3. Aho, A.V.: Indexed grammars – an extension of context-free grammars. J. ACM 15(4), 647–671 (1968)
4. Barendregt, H.: The type free lambda calculus. In: Handbook of Mathematical Logic, ch. D.7, pp. 1091–1132. North-Holland (1977)

5. Barendregt, H., Klop, J.W.: Applications of infinitary lambda calculus. Inf. Comput. 207(5), 559–582 (2009)
6. Cachat, T., Walukiewicz, I.: The Complexity of Games on Higher Order Pushdown Automata. Internal report
7. Carayol, A., Wöhrle, S.: The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 112–123. Springer, Heidelberg (2003)
8. Caucal, D.: On Infinite Terms Having a Decidable Monadic Theory. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 165–176. Springer, Heidelberg (2002)
9. Courcelle, B.: A representation of trees by languages I. Theor. Comput. Sci. 6, 255–279 (1978)
10. Courcelle, B.: Recursive applicative program schemes. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), pp. 459–492. Elesvier (1990)
11. Courcelle, B., Nivat, M.: Algebraic families of interpretations. In: FOCS (1976)
12. Courcelle, B., Walukiewicz, I.: Monadic second-order logic, graphs and unfoldings of transition systems. Annals of Pure and Applied Logic 92, 35–62 (1998)
13. Damm, W.: The IO– and OI–hierarchies. Theoretical Computer Science 20(2), 95–208 (1982)
14. Dezani-Ciancaglini, M., Giovannetti, E., de' Liguoro, U.: Intersection Types, Lambda-models and Böhm Trees. In: MSJ-Memoir "Theories of Types and Proofs", vol. 2, pp. 45–97. Mathematical Society of Japan (1998)
15. Engelfriet, J.: Iterated push-down automata and complexity classes. In: 15th STOC 1983, pp. 365–373 (1983)
16. Engelfriet, J., Schmidt, E.: IO and OI. Journal of Computer and System Sciences 15(3), 328–353 (1977)
17. Hague, M., Murawski, A.S., Ong, C.-H.L., Serre, O.: Collapsible pushdown automata and recursion schemes. In: LICS 2008, pp. 452–461. IEEE Computer Society (2008)
18. Ianov, Y.: The logical schemes of algorithms. In: Problems of Cybernetics I, pp. 82–140. Pergamon, Oxford (1969)
19. Indermark, K.: Schemes with Recursion on Higher Types. In: Mazurkiewicz, A. (ed.) MFCS 1976. LNCS, vol. 45, pp. 352–358. Springer, Heidelberg (1976)
20. Jancar, P.: Decidability of DPDA language equivalence via first-order grammars. In: LICS 2012 (2012)
21. Knapik, T., Niwiński, D., Urzyczyn, P.: Higher-Order Pushdown Trees Are Easy. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 205–222. Springer, Heidelberg (2002)
22. Knapik, T., Niwiński, D., Urzyczyn, P., Walukiewicz, I.: Unsafe Grammars and Panic Automata. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1450–1461. Springer, Heidelberg (2005)
23. Kobayashi, N.: Types and higher-order recursion schemes for verification of higher-order programs. In: POPL 2009, pp. 416–428. ACM (2009)
24. Kobayashi, N.: Higher-order model checking: From theory to practice. In: LICS 2011, pp. 219–224 (2011)
25. Kobayashi, N., Ong, C.-H.L.: Complexity of model checking recursion schemes for fragments of the modal mu-calculus. Logical Methods in Computer Science 7(4) (2011)

26. Kobayashi, N., Ong, L.: A type system equivalent to modal mu-calculus model checking of recursion schemes. In: LICS 2009, pp. 179–188 (2009)
27. Manna, Z.: Mathematical Theory of Computation. McGraw-Hill (1974)
28. Maslov, A.: The hierarchy of indexed languages of an arbitrary level. Soviet. Math. Doklady 15, 1170–1174 (1974)
29. Maslov, A.: Multilevel stack automata. Problems of Information Transmission 12, 38–43 (1976)
30. Milner, R.: Models of LCF. Memo AIM-186. Stanford University (1973)
31. Nivat, M.: On interpretation of recursive program schemes. In: Symposia Mathematica, vol. 15 (1975)
32. Ong, C.-H.L.: On model-checking trees generated by higher-order recursion schemes. In: LICS 2006, pp. 81–90 (2006)
33. Parys, P.: On the significance of the collapse operation. In: LICS 2012 (2012)
34. Plotkin, G.D.: LCF considered as a programming language. Theor. Comput. Sci. 5(3), 223–255 (1977)
35. Salvati, S.: Recognizability in the Simply Typed Lambda-Calculus. In: Ono, H., Kanazawa, M., de Queiroz, R. (eds.) WoLLIC 2009. LNCS, vol. 5514, pp. 48–60. Springer, Heidelberg (2009)
36. Salvati, S., Manzonetto, G., Gehrke, M., Barendregt, H.: Loader and Urzyczyn Are Logically Related. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 364–376. Springer, Heidelberg (2012)
37. Salvati, S., Walukiewicz, I.: Krivine Machines and Higher-Order Schemes. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 162–173. Springer, Heidelberg (2011)
38. Scott, D.: Continuous lattices. In: Proc. of Dalhousie Conference. Lecture Notes in Mathematics, vol. 188, pp. 311–366. Springer (1972)
39. Sénizergues, G.: L(A)=L(B)? Decidability results from complete formal systems. Theor. Comput. Sci. 251(1-2), 1–166 (2001)
40. Sénizergues, G.: L(A)=L(B)? A simplified decidability proof. Theor. Comput. Sci. 281(1-2), 555–608 (2002)
41. Stirling, C.: Deciding DPDA Equivalence Is Primitive Recursive. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 821–832. Springer, Heidelberg (2002)

Transportation under Nasty Side Constraints

Gerhard J. Woeginger

Department of Mathematics and Computer Science
TU Eindhoven, Netherlands
gwoegi@win.tue.nl

Abstract. The talk discusses planning problems where a set of items has to be transported from location A to location B subject to certain collision and/or resource constraints. We analyze the behavior of these problems, discuss their history, and derive some of their combinatorial and algorithmic properties.

The first transportation problem. Let V be a set of items/vertices, and let $G = (V, E)$ be a graph. We consider a scenario where the items in V have to be transported from point A to point B. There is a transportation device with enough capacity to carry $b \geq 1$ of the items, and there is a single driver. If two items are connected by an edge in E, they are *conflicting* and thus cannot be left alone together without human supervision. A *feasible schedule* is a finite sequence of triples (A_1, T_1, B_1), (A_2, T_2, B_2), $\ldots, (A_s, T_s, B_s)$ of subsets of the item set V that satisfies the following conditions (FS1)–(FS3). The odd integer s is called the *length* of the schedule.

(FS1) For every k, the sets A_k, T_k, B_k form a partition of V. The sets A_k and B_k form stable sets in G. The set T_k contains at most b elements.

(FS2) The sequence starts with $A_1 \cup T_1 = V$ and $B_1 = \emptyset$, and the sequence ends with $A_s = \emptyset$ and $T_s \cup B_s = V$.

(FS3) For even $k \geq 2$, we have $T_k \cup B_k = T_{k-1} \cup B_{k-1}$ and $A_k = A_{k-1}$. For odd $k \geq 3$, we have $A_k \cup T_k = A_{k-1} \cup T_{k-1}$ and $B_k = B_{k-1}$.

Intuitively speaking, the kth triple encodes the kth trip: A_k contains the items currently in point A, T_k the items that are currently transported, and B_k the items in point B. Odd indices correspond to forward trips, and even indices correspond to backward trips. Condition (FS1) states that the (unsupervised) sets A_k and B_k must not contain conflicting item pairs, and that set T_k must fit into the transportation device. Condition (FS2) concerns the first trip and the final trip. Condition (FS3) says that whenever the man reaches point A or B, he may arbitrarily re-divide the set of available items.

We are interested in the smallest possible capacity of a transportation device for which a given graph $G = (V, E)$ possesses a feasible schedule. For instance for the path P_3 on three vertices, it can be seen that a capacity $b = 1$ is sufficient. We discuss a variety of combinatorial and algorithmical results on these concepts; in particular we show that the smallest possible capacity has an NP-certificate.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 61–62, 2012.

The second transportation problem. Let I be a set of items, and let $w(i)$ be the positive integer weight of item $i \in I$. For $J \subseteq I$ we throughout denote $w(J) = \sum_{j \in J} w(j)$, and as usual we let $w(\emptyset) = 0$. We consider a scenario where the items have to be transported from a point A at the top of a building to a point B at the bottom of the building. The transporation is done with the help of a pulley with a rope around it, and a basket fastened to each end of the rope of equal weight. One basket coming down would naturally draw the other basket up. To keep the system stable, the weights of the item sets in the two baskets must not differ by more than a given threshold Δ.

A *state* of the underlying discrete system is specified by the item set $J \subseteq I$ that currently is at point A, and with the remaining items in $I - J$ located at point B. The system can move directly from state $J \subseteq I$ to state $K \subseteq I$ if

$$|w(J \cap (I - K)) - w(K \cap (I - J))| \;\le\; \Delta,$$

where the positive integer bound Δ specifies the maximum allowed weight difference between the two exchanged subsets in the baskets. A state K is *reachable* from state J, if there is a sequence of moves that transforms J into K. It is easy to see that reachability is a symmetric relation.

We are interested in the following question: Given an item set I with weights $w(i)$, a positive integer bound Δ, an initial state I_0, and a final state I_1. Is the goal state I_1 reachable from the initial state I_0? We discuss a number of results on the algorithmic and combinatorial behavior of this motion planning problem. In particular, we show that it is Π_2^p-complete. The special case where the item weights are encoded in unary is (trivially) solvable in pseudo-polynomial time. The special case where the number of moves is bounded by a number encoded in unary is NP-complete. Some other natural (hevaily structured) special cases turn out to be solvable in polynomial time.

References

1. Csorba, P., Hurkens, C.A.J., Woeginger, G.J.: The Alcuin number of a graph and its connections to the vertex cover number. SIAM Journal on Discrete Mathematics 24, 757–769 (2010)
2. Eggermont, C., Woeginger, G.J.: Motion planning with pulley, rope, and baskets. In: Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012). Leibniz International Proceedings in Informatics, vol. 14, pp. 374–383 (2012)

Computation of Least Fixed Points

Mihalis Yannakakis

Department of Computer Science
Columbia University
mihalis@cs.columbia.edu

Many problems from different areas can be formulated as problems of computing a fixed point of a suitable function. Classical examples include the computation of equilibria for games, price equilibria for markets, and many others. There has been significant progress in understanding better the computational nature of such problems and characterizing their complexity in terms of classes like FIXP, which captures the complexity of the computation of fixed points for general (nonlinear) algebraic functions, with the 3-player Nash equilibrium problem as a prototypical example, and PPAD for the computation of fixed points for piecewise linear functions, with the 2-player Nash equilibrium problem as a prototypical example.

For many problems, the function in the fixed point formulation is *monotone*, and the objects we want to compute are given by a specific fixed point, namely the *least fixed point* of the function. Many models and problems from a broad variety of areas can be thus expressed as least fixed point computation problems, including in particular the analysis of various probabilistic models, problems in stochastic optimization, and games. Examples include the analysis of multitype branching processes; stochastic context-free grammars; recursive Markov chains; quasi-birth-death processes; (recursive) Markov decision processes; stochastic games, whether turn-based or concurrent, flat finite-state or recursive. The objects of interest in all these problems can be computed by a natural iterative algorithm based on the least fixed point formulation, however, the algorithm takes in all these cases exponential time to converge. The question is whether the desired objects can be computed in polynomial time by alternative methods. In recent years there has been significant progress in answering this question for several of these problems. It turns out that for some classes of functions we can compute the least fixed point in polynomial time and thus we can analyze efficiently several of these models, while for others there are strong indications that they cannot be solved in polynomial time, and for yet others the question remains open. In this talk we will discuss progress in this area. The talk is based on a series of works with Kousha Etessami and Alistair Stewart.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, p. 63, 2012.

Unordered Constraint Satisfaction Games

Lauri Ahlroth and Pekka Orponen

Department of Information and Computer Science and
Helsinki Institute for Information Technology HIIT
Aalto University, Finland
{Lauri.Ahlroth,Pekka.Orponen}@aalto.fi

Abstract. We consider two-player constraint satisfaction games on systems of Boolean constraints, in which the players take turns in selecting one of the available variables and setting it to *true* or *false*, with the goal of maximising (for Player I) or minimising (for Player II) the number of satisfied constraints. Unlike in standard QBF-type variable assignment games, we impose *no order* in which the variables are to be played. This makes the game setup more natural, but also more challenging to control. We provide polynomial-time, constant-factor approximation strategies for Player I when the constraints are parity functions or threshold functions with a threshold that is small compared to the arity of the constraints. Also, we prove that the problem of determining if Player I can satisfy all constraints is PSPACE-complete even in this unordered setting, and when the constraints are disjunctions of at most 6 literals (an unordered-game analogue of 6-QBF).

1 Introduction

An instance of a *constraint satisfaction problem* (CSP) comprises a set of m non-constant Boolean functions $C = \{c_1, \ldots, c_m\}$ over a common set of n variables $X = \{x_1, \ldots, x_n\}$. In this work we only consider *Boolean* CSP's, so that each constraint $c \in C$ is a mapping $c : \{0,1\}^n \to \{0,1\}$. (The truth values *false*, *true* are identified with the integers $0, 1$ as usual.) A constraint c is *satisfied* by a truth assignment $x \in \{0,1\}^n$ if $c(x) = 1$. The most commonly studied versions of CSPs ask whether all of the given constraints can be satisfied simultaneously (typically an NP-complete problem), or if not, then what is the fraction of constraints that can be satisfied by a polynomial-time approximation algorithm.

Another perspective is to consider the *combinatorial games* [12,18] defined by CSP instances. Here two players, I and II, take turns in setting values of the variables in X, with Player I's goal being to eventually satisfy as many of the constraints in C as possible, and Player II's goal being the opposite. In this framework one can again ask if Player I has a *(comprehensive) winning strategy*, i.e. whether she can satisfy *all* the constraints, no matter what Player II does. Or if a winning strategy does not exist or is hard to compute, then what is the fraction of constraints that Player I can satisfy by a polynomial-time computable approximate strategy against Player II.

Such *constraint satisfaction games* have applications in e.g. formal methods [1,17] and adversarial planning [2,3], but are of course also intrinsically interesting. A paradigmatic example is QBF, which can be viewed as a two-player version of Satisfiability,

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 64–75, 2012.

where the players take turns in assigning values to the variables in a prespecified order [18]. As is well-known, QBF is PSPACE-complete [19], similarly to many other terminating two-player games [10]. Recently there have also been major developments in developing a taxonomy of such quantified constraint satisfaction games, relating their complexity to the structure of the constraints involved [6,7,16]. This work follows the example of the quite comprehensive taxonomy on the approximability of CSPs according to the structure of their defining constraints, as achieved in [15].

Existing results on the complexity of constraint satisfaction games assume players assign values to the variables in a *predefined (quantification) order*. This is a somewhat unnatural restriction if one views CSPs in the general framework of combinatorial games. An unordered setting arises naturally e.g. in the *colouring construction game* introduced in [5]. In this game the players take turns in colouring the vertices of a graph, with Player I trying to achieve a proper colouring of the full graph and Player II attempting to foil her. In the case of two colours, this is a Boolean CSP, and the complexity of this game (i.e. determining the existence of a comprehensive winning strategy for Player I) is indicated in [5] as an interesting open problem.

The assumption of a fixed variable order also leads to an overly pessimistic view on the approximability of games of this type. For instance, the two published studies [9,13] addressing the approximability of (ordered) constraint satisfaction games in the sense of maximising the number of constraints won by Player I both present only *negative* results on the existence of approximate strategies. In particular, both of these papers show that for some $\varepsilon > 0$, MAX-QBF is PSPACE-hard to approximate within $1-\varepsilon$ times optimum. Articles [4,8] consider the somewhat different task of optimising a supplementary objective function across the comprehensive winning strategies for Player I.

A symptom of the unnaturality of variable-ordered games is that they effectively allow one of the players to make many consecutive moves, by forcing the other player to play dummy variables that do not affect the final outcome. In the extreme case, one can e.g. have *all* the even-indexed variables be dummies, so that the outcome is totally indifferent to the moves of Player II. By allowing a free choice of variables neither player can be discriminated in this way, and for this reason the unordered game has a more interesting structure than standard QBF, both as a game and in the sense of approximability.

In this paper we wish to initiate the study of such *unordered* constraint satisfaction games and their approximate strategies. We first define the model in Section 2. In the subsequent Sections we establish our *positive* results on the approximability of unordered constraint satisfaction games. In Section 3 we show that in a game where the constraints are parity functions (linear equations mod 2) with bounded variable co-occurrences, Player I can always win close to $\frac{1}{2}$ of the constraints. In Section 4 we show that if the constraints are threshold functions of arity k and the threshold is $\mu = O\left(\frac{k}{\log k}\right)$, then Player I can win a fraction of the clauses which approaches 1 exponentially as k increases. A weaker result holds when μ satisfies merely $\mu \leq \frac{k}{2}$. Note that the strong version of the result applies in particular to games on disjunctive constraints, since disjunctions are threshold functions with $\mu = 0$.

In Section 5 we establish our fundamental *negative* result showing that also the unordered game analogue of the QBF problem, the Game on Boolean Formulas (GBF), is PSPACE-complete, even when the constraints are disjunctions of at most 6 literals. We conclude in Section 6 with a summary and suggestions for some further research directions.

2 The Game on Boolean Formulas

Our generic example of an unordered constraint satisfaction game is the *Game on Boolean Formulas (GBF)*. An instance of this game is given by a set of m non-constant Boolean formulas $C = \{c_1, \ldots, c_m\}$ over a common set of n variables $X = \{x_1, \ldots, x_n\}$. We refer to the formulas in C as *clauses* even though we do not in general require them to be disjunctions. If all the clauses are disjunctions of *width* $\leq k$, i.e. with at most k literals per clause, then we refer to the game as k-GBF.

A game on C proceeds so that on each turn the player to move selects one of the previously nonselected variables and assigns a truth value to it. Player I starts, and the game ends when all variables have been assigned a value. In the decision version of GBF, the question is whether Player I has a comprehensive winning strategy, by which she can make all clauses satisfied no matter what Player II does. In the positive case we say that the instance is *GBF-satisfiable*. In the maximisation version MAX-GBF, the objective of Player I is to maximise the number of satisfied clauses $m_t := |\{c \in C \mid c(x_1, x_2, \ldots x_n) = 1\}|$, and the objective of Player II is to maximise the number $m_f := m - m_t$ of unsatisfied clauses, making this a zero-sum game.

Recall that a Boolean formula $P(x_1, x_2, \ldots x_n)$ is QBF-satisfiable if Player I can ensure that P becomes satisfied when the players alternate selecting values to $x_1, x_2, \ldots x_n$ *in the given order*. The standard (MAX-)QBF problem can be viewed as a version of (MAX-)GBF, with the additional requirement that the variables must be played in the predetermined order $x_1, x_2, \ldots, x_n$.

3 An Approximate Strategy for Even-Odd Games

In an *even-odd game* the GBF clauses $c \in C$ are of the form $\sum_{i \in J_c} x_i \equiv d_c (\text{mod } 2)$, where for each $c \in C$, $J_c \subseteq \{1, \ldots, n\}$ and $d_c = 0$ or $d_c = 1$. Without loss of generality one can assume that there are no negative literals.

Theorem 1. *In an even-odd game on n variables and m clauses, Player I has a polynomial-time strategy that secures her at least a fraction of* $\frac{1}{2} - \frac{n\delta}{4m}$ *of all clauses, where* $\delta = \max_{i \neq i'} |\{c \mid x_i \in c, x_{i'} \in c\}|$.

Proof. For $1 \leq i \leq \frac{n}{2}$, denote the number of clauses completing as *true* (resp. *false*) due to Player I's ith move as α_i ($\bar{\alpha}_i$) and completing as *true* (*false*) due to Player II's ith move as β_i ($\bar{\beta}_i$). A clause *completes* when its last variable is set to a value. The strategy for Player I is simply to always make a move maximising the difference $\alpha_i - \bar{\alpha}_i$.

When Player I is selecting the variable for her ith move, the variable y selected by Player II on his ith move is also available. Player I's ith move creates at most δ new opportunities for II to complete clauses with y, so by the strategy of I it must be the case that

$$\alpha_i - \bar{\alpha}_i \geq \bar{\beta}_i - \beta_i - \delta,$$

which can be rearranged as

$$\bar{\alpha}_i + \bar{\beta}_i \leq \alpha_i + \beta_i + \delta.$$

Now the total number of clauses has the upper bound

$$m = \sum_i (\alpha_i + \beta_i + \bar{\alpha}_i + \bar{\beta}_i) \leq 2\sum_i (\alpha_i + \beta_i) + \frac{n}{2}\delta = 2m_t + \frac{n}{2}\delta.$$

Hence the fraction of satisfied clauses is at least $\frac{m_t}{m} \geq \frac{1}{2} - \frac{n\delta}{4m}$.

Corollary 1. *In an even-odd game where $\delta \frac{n}{m} \leq \gamma$ for a constant $\gamma < 2$, Player I can secure a constant fraction of all clauses.*

4 Approximate Strategies for Threshold Games

In a *μ-threshold game*, the clauses c are subsets of literals from $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$, and a clause $c \in C$ is satisfied if and only if it contains at least $\mu + 1$ true literals $l \in c$. We assume that each clause has at least k literals, that no literal occurs multiple times in the same clause, and that no literal co-occurs with its negation in the same clause. The case $\mu = 0$ corresponds to disjunctive clauses. When all clauses are exactly of length k and $\mu + 1 = k$, the clauses correspond to conjunctions.

In the following, we present an efficient approximate strategy for Player I in the case of small values of μ. The potential function approach used to drive the strategy is inspired by the MAX-SAT approximation algorithm presented in [14], even though the details are very different.

Theorem 2. *Let $\mu \in \mathbf{N}$. In a μ-threshold game on clauses of size at least k, Player I has a polynomial-time strategy that secures her at least a fraction $1 - O(\frac{k^\mu}{2^{k/2}})$ of all clauses.*

To describe the strategy indicated in Theorem 2 we need some more notation. Given a clause c, denote the number of unset literals present in c by $f(c)$, and the number of literals already set to *true* by $t(c)$. Define q as the positive root of $q^2 + q/k - 2 = 0$, which gives

$$\sqrt{2}\frac{1}{1 + 1/(k\sqrt{2})} < q < \sqrt{2}. \tag{1}$$

The *weight* of a clause is defined as

$$u(c) := \begin{cases} q^{-f(c)}k^{-t(c)}, \text{ if } t(c) \leq \mu \\ 0, \text{ else.} \end{cases}$$

Also, the *weight* of a literal l is $u(l) := \sum_{l \in c} u(c)$, and the *potential* of a partially played game is $U := \sum_{c \in C} u(c)$.

Proof (Theorem 2).

Player I's strategy is to always find a literal l of maximum weight $u(l)$ and set l to *true*. We shall prove that this strategy ensures that U is nonincreasing over each pair of moves by the two players.

Note first that if Player I sets literal x to *true* and Player II sets literal y to *true* then $u(x) \geq \max\{u(y), u(\bar{x}), u(\bar{y})\}$.

Over one move the potential changes by $\Delta_x U \leq (q-1)u(\bar{x}) + (q/k-1)u(x)$. After this move the weights change to $u'(c)$, but the new weights still satisfy $u'(c) \leq qu(c)$ clause-wise, implying

$$\Delta_y U \leq (q-1)qu(\bar{y}) + (q/k-1)u'(y) \leq (q^2-q)u(x).$$

Hence

$$\Delta_x + \Delta_y \leq (q-1+q/k-1+q^2-q)u(x) = (q^2+q/k-2)u(x) = 0.$$

This and $\Delta_x \leq 0$ imply that U is nonincreasing over the whole game.

From Eq. (1) one obtains $q^k \geq \sqrt{2}^k e^{-1/\sqrt{2}}$ and further

$$\frac{k^\mu}{q^k} \leq \frac{k^\mu}{2^{k/2}} e^{1/\sqrt{2}}.$$

Now

$$k^\mu m_f \leq U_{end} \leq U_{begin} \leq mq^{-k},$$

implying the final claim since

$$\frac{m_f}{m} \leq \frac{k^\mu}{q^k} \leq \frac{k^\mu}{2^{k/2}} e^{1/\sqrt{2}}.$$

Corollary 2. *In a μ-threshold game where $\frac{k}{\log k}\frac{\log 2}{2} - \frac{1}{\sqrt{2}\log k} - \mu \geq \gamma$ for a constant $\gamma > 0$, Player I can secure a constant fraction of all clauses.*

Theorem 2 gives a positive bound up to $\mu \leq \mu^*$ for some $\mu^* = \theta\left(\frac{k}{\log k}\right)$. For large threshold values $\mu = \Omega\left(\frac{k}{\log k}\right)$ one can apply an elementary algorithm to improve the bound.

Theorem 3. *Let $\mu \in \mathbf{N}$. In a μ-threshold game on clauses of size at least k, Player I has a polynomial-time strategy that secures her at least a fraction of $\frac{k-2\mu}{2k-2\mu}$ of all clauses.*

Proof. Any literal l occurs in $v(l) = |\{c \in C \mid l \in c\}|$ clauses, and we define the weight of a literal as the difference $u(l) := v(l) - v(\bar{l})$. The strategy is to always find a literal l of maximum weight $u(l)$ and set l to *true*.

Consider the numbers of *true* and *false* literal occurrences

$$T = |\{(c,l) \mid c \in C,\ l \in c,\ l\ true\}|,$$
$$F = |\{(c,l) \mid c \in C,\ l \in c,\ l\ false\}|.$$

The strategy used by Player I always targets a literal with maximum weight. As the moves never alter the weights of the other literals, the sum of weights of the literals set to *true* is always nonnegative. Especially this holds at end of the game. Hence $T \geq F$, and

$T \geq \frac{m}{2}k$. As each unsatisfied clause contains at most μ *true* literals, T can be bounded as

$$\frac{m_t + m_f}{2} k \leq T \leq \mu m_f + k m_t.$$

Rearranging yields $\frac{m_t}{m_f} \geq \frac{k-2\mu}{k}$ and finally

$$\frac{m_t}{m} \geq \frac{k-2\mu}{2k-2\mu} = \frac{1}{2} - \frac{\mu}{2(k-\mu)}.$$

The result of Theorem 3 yields a positive lower bound on the number of clauses won by Player I up to a threshold of $\mu < \frac{k}{2}$. Actually no strategy can guarantee a positive fraction of the clauses to her if $\mu \geq \frac{k}{2}$. To verify this, consider an instance that has different variables in each clause, and Player II always responds by playing a literal *false* in the same clause where Player I just played. Independent of Player I's strategy, all clauses end up *false*. Naturally, this observation does not preclude approximation bounds with respect to the value of an optimal solution, or positive results under more structural assumptions on the instance.

5 PSPACE-Completeness of the GBF Problem

We now prove that the decision problem version of GBF is PSPACE-complete. We present two versions of this result. Theorem 4 is proved by a direct but nontrivial reduction from QBF. However since this reduction yields a GBF instance with only a *single* constraint formula, we present also a second version, Theorem 5, which has a more complex proof but yields a constraint system consisting of disjunctions of width ≤ 6.

Theorem 4. *The problem of deciding GBF-satisfiability of a Boolean formula is PSPACE-complete.*

Proof. The existence of a winning strategy for Player I can clearly be determined in polynomial space by a systematic depth-first min-max search of all the play sequences.

To show PSPACE-hardness we design a reduction from QBF satisfiability. Fix a QBF instance, which consists of a Boolean formula $P(x_1, x_2 \ldots x_n)$. We construct a Boolean formula $\tilde{P}$ where Player I has a GBF winning strategy if and only if P is QBF-satisfiable. The instance $\tilde{P}$ is going to be essentially P embedded with gadgets that force the players to respect the predefined order of variables.

We present explicitly a gadget that requires Player I to start from playing x_i. Given a Boolean formula $R(x_1, \ldots x_n)$, consider a GBF with the formula

$$\begin{aligned} &R^i(a_i, b_i, c_i, d_i, e_i, x_1, \ldots x_{i-1}, x_{i+1}, \ldots x_n) := \\ &[R(x_1, \ldots x_{i-1}, a_i, x_{i+1}, \ldots x_n) \wedge e_i \wedge (a_i \vee c_i) \wedge (b_i \vee d_i)] \vee (a_i \wedge d_i) \vee (c_i \wedge b_i). \end{aligned} \tag{2}$$

For sake of presentation we omit the subscript i from the variables $S_i = \{a_i, b_i, c_i, d_i, e_i\}$.

The idea behind the construction is to have two copies of the variable x_i that end up with equal values, $a = b$. Also the negation $\overline{x}_i$ is represented by two variables, $c = d \neq b$. e is a dummy variable that switches turn. The structure of the gadget guarantees that after Player I fixes a value to x_i, the players continue filling the remaining gadget variables $S_i = \{a,b,c,d,e\}$ accordingly. A deviation from this guideline leads to a quick loss for the deviating player. Also, Player I cannot postpone the decision on the value of variable x_i to a later point without losing the game.

Let us define a generalised version of GBF imposing some restrictions on the order of variables. In a *GBF under queue* $q = (x_{i_1}, x_{i_2}, \dots x_{i_r})$, $0 \le r \le n$, the jth move must be done on the queue variable x_{i_j} as long as $j \le r$. For $j > r$ the variables can be chosen freely among the unset variables as in standard GBF. Also, define a concatenation of queues as $q \circ q' = (x_{i_1}, \dots x_{i_r}, x'_{i_1}, \dots x'_{i'_r})$.

Lemma 1. *Player I has a GBF winning strategy on R^i if and only if Player I has a GBF winning strategy on R under queue (x_i).*

Proof. The proof needs some tedious case-by-case analysis. To prune out obvious inferior moves on R^i, note that the gadget has no negated variables. We observe that Player I is always better off by playing a value *true* than a value *false* in the gadget variables S_i. The opposite holds for Player II. Further, let us encode game positions by concatenating the variables selected by the players, with # denoting any variable outside S_i. For example, the game that started with moves $a = 1$; $x_2 = 0$; $e = 1$; $d = 0$ in this order would be compressed as $a\overline{\#}e\overline{d}$. We also use $*$ as a wild character with no set restrictions.

Consider all possible first player moves.

Case a**:** If Player I starts with a, the threat of $a * d$ forces Player II to continue as $a\overline{d}$. Now, the threat of $a\overline{d} * \overline{b}$ forces Player I to continue as $a\overline{d}b$. The threat $a\overline{d}b * c$ enforces $a\overline{d}b\overline{c}$ and Player I is forced to continue to $a\overline{d}b\overline{c}e$. Variables in S get fixed and the game continues with Player II's turn on the truncated game $R_{i,1} = R(x_1, \dots x_{i-1}, 1, x_{i+1}, \dots x_n)$.

Case b**:** As Case a with swapping variables as $a \leftrightarrow b$, $c \leftrightarrow d$. Variables in S_i get fixed and the game continues with Player II's turn on $R_{i,1}$.

Case c**:** As Case a with swapping variables as $a \leftrightarrow c$, $b \leftrightarrow d$. Variables in S_i get fixed and the game continues with Player II's turn on $R_{i,0} = R(x_1, \dots x_{i-1}, 0, x_{i+1}, \dots x_n)$.

Case d**:** As Case a with swapping variables as $a \leftrightarrow d$, $b \leftrightarrow c$. Variables in S_i get fixed and the game continues with Player II's turn on $R_{i,0}$.

Case e**:** Player II *can* continue to $e\overline{a}$. $e\overline{a} * \overline{c}$ would be a loss for Player I, hence the third move must lead to $e\overline{a}c$. Player II can reply by $e\overline{a}c\overline{b}$, which enforces $e\overline{a}c\overline{b}d$. Variables in S_i are fixed and the game continues with Player II's turn on $R_{i,0}$. For $e\overline{c}, e\overline{b}$ and ed the analysis is similar with variable swaps, leading to fixed S_i and Player II's turn on $R_{i,0}$. Effectively, Player II decides value for x_i on game R and keeps turn - this is no better for Player I than Case a.

Case #: As Case e, with the exception of sixth move being $\bar{e}$, yielding a loss for Player I.

Collecting the case-by-case results, we conclude that Player I has a winning strategy on R' if and only if Player I wins either of the subgames $R_{i,0}$, $R_{i,1}$ with Player II to move first. This condition is equivalent to Player I having a winning strategy on R that first assigns a value to x_i, and the proof is finished.

The lemma easily yields the following corollaries.

Corollary 3. *Let $i \neq j$. Given a Boolean formula $R(x_1,\dots x_n)$, Player I has a winning strategy on $(\neg(\neg R)^j)^i$ if and only if Player I has a GBF winning strategy on R under queue (x_i,x_j).*

Proof. By a simple role reversal observation Player I has a winning strategy in a game Q if and only if I has no winning strategy in the game $\neg Q$ that is started by Player II. By Lemma 1 and role reversal I has a winning strategy on $(\neg(\neg R)^j)^i$ if and only if I has no winning strategy on at least one of the subgames $(\neg R_{i,0})^j$, $(\neg R_{i,1})^j$. Applying the lemma and role reversal again, the previous is equivalent to Player I winning either both $R_{i,0;j,0}$ and $R_{i,0;j,1}$, or both $R_{i,1;j,0}$ and $R_{i,1;j,1}$. But this is just the condition that Player I has a GBF winning strategy on R with requirements of Player I playing first x_i and Player II playing x_j immediately thereafter.

Corollary 4. *Let $i \neq j$, $x_i,x_j \notin q$, $|q|$ even. Given a Boolean formula $R(x_1,\dots x_n)$, Player I has a winning strategy on $(\neg(\neg R)^j)^i$ under q if and only if Player I has a GBF winning strategy on R under queue $q \circ (x_i,x_j)$.*

Proof. Assume there is a winning strategy for either of the cases. This is easily converted to a winning strategy in the other: simply play q as the winning strategy suggests. Consider each of the subgames after q has been played. One of them is known to have a winning strategy, whence by Corollary 3 there must be a winning strategy for the other as well.

Corollary 5. *Given a Boolean formula $P(x_1 \dots x_n)$, there exists a polynomial-time computable Boolean formula $\tilde{P}(y_1 \dots y_{5n})$ such that P is QBF-satisfiable if and only if $\tilde{P}$ is GBF-satisfiable.*

Proof. Consider

$$\tilde{P} = (\neg(\neg \dots (\neg(\neg P)^{n'})^{n'-1} \dots)^2)^1,$$

where $n' = 2\lfloor \frac{n}{2} \rfloor$. Using Corollary 4 one can transform two outermost gadgets into a queue with winning strategies staying equivalent. Repeated use $\frac{n'}{2}$ times implies that existence of a winning strategy in $\tilde{P}$ is equivalent to existence of a winning strategy in P under queue $q = (x_1,x_2,\dots,x_{n'})$. But this is the same game as QBF play on P with order $x_1,x_2,\dots x_n$.

PSPACE-hardness of the GBF-satisfiability for Boolean formulas follows now directly from Corollary 5.

Let us then consider the more restricted GBF instances where the constraint system is in k-conjunctive normal form.

Theorem 5. *Deciding GBF-satisfiability of a system of disjunctions of width ≤ 6 is PSPACE-complete.*

Proof. As in Theorem 4, it is straightforward to see that the problem is in PSPACE.

To establish the PSPACE-hardness of the problem we outline a reduction from the PSPACE-complete *Shannon switching game on vertices (SSG)* [11]. As we shall see, in fact any PSPACE-complete game with an unordered set of binary game variables and a polynomial-time winning predicate would do as a basis for the reduction, but let us for concreteness focus here on the SSG.

An SSG instance is given by an undirected graph $G = (V,E)$ with two distinguished vertices $s,t \in V$. The players take turns in selecting vertices in the graph, excluding s and t, and Player I wins if she manages to establish an $s-t$ path in G through vertices owned by her. Player II wins if he can keep this from happening until all the vertices have been played. As proved in [11], it is PSPACE-complete to determine if in a given graph (G,s,t) Player I has a winning strategy.

We now show how to effectively construct from an SSG instance (G,s,t) a GBF instance (X,C), where all the clauses in C are disjunctions with at most 6 literals per clause, and Player I has a winning strategy on (G,s,t) if and only if she has a winning strategy on (X,C).

The construction is completely general, and only depends on the high-level characteristics of the SSG game, as indicated earlier. Let the graph G have n vertices, excluding s and t, and let $x_1,\ldots,x_n$ be binary variables indicating whether each vertex i is selected by Player I ($x_i = 1$) or by Player II ($x_i = 0$). Let $W(x_1,\ldots,x_n)$ be a polynomial-time computable predicate indicating whether the resulting configuration is a win for Player I ($W(x) = 1$) or for Player II ($W(x) = 0$).

For a given SSG instance (G,s,t), the predicate $W(x)$ can be effectively expanded into a polynomial-size circuit with, say, p gates. Since $\text{NAND}(x,y) = \overline{x \wedge y} = \bar{x} \vee \bar{y}$ is a universal basis operation for Boolean circuits, we for simplicity and w.l.o.g. assume that all the gates are NAND gates. Let the gates of the W circuit be topologically sorted into a sequence $(g_1,g_2,\ldots,g_p)$, so that each gate g_k computes an intermediate result $z_k \leftarrow \overline{y}_i \vee \overline{y}_j$, where y_i (resp. y_j) is either z_i for $i < k$ (resp. z_j for $j < k$) or one of the input variables $x_1,\ldots,x_n$, and $z_p = 1$ iff $W(x) = 1$.

Let now the variables of the corresponding GBF instance (X,C) be the original input variables $x_1,\ldots,x_n$, together with $2p$ auxiliary variables $z_1,\ldots,z_p$, $a_1,\ldots,a_p$. For each gate g_k of the form $z_k \leftarrow \bar{z}_i \vee \bar{z}_j$ in the W circuit, we introduce a clause

$$c_k \quad = \quad ((z_k \equiv \bar{z}_i \vee \bar{z}_j) \vee (z_k \equiv a_k) \vee (z_i \equiv a_i) \vee (z_j \equiv a_j)).$$

For gates where one or both of the argument literals are input variables, the clauses are similar except that the disjuncts of the form "$x \equiv a$" are omitted for an input variable x.

We observe first that because each clause c_k refers to at most 6 variables, when it is expanded into conjunctive normal form, it expands into some bounded number $d \leq 3^6$ of disjunctions ("miniclauses") $c_{k1},\ldots,c_{kd}$ with at most 6 literals per miniclause, satisfying $c_k \equiv \bigwedge_{l=1}^{d} c_{kl}$.[1] Altogether these miniclauses thus satisfy

$$\bigwedge_{k=1}^{p} c_k \quad \equiv \quad \bigwedge_{k=1}^{p} \bigwedge_{l=1}^{d} c_{kl},$$

and we take as the set of clauses C in our (6-)GBF instance $C := \{c_{kl} \mid k = 1,\ldots,p, l = 1,\ldots,d\}$.

[1] A more careful analysis shows that in fact $c_k \equiv (z_i \vee z_j \vee z_k \vee \bar{a}_i \vee \bar{a}_j \vee \bar{a}_k) \wedge (\bar{z}_i \vee z_j \vee z_k \vee a_i \vee \bar{a}_j \vee \bar{a}_k) \wedge (z_i \vee \bar{z}_j \vee z_k \vee \bar{a}_i \vee a_j \vee \bar{a}_k) \wedge (\bar{z}_i \vee \bar{z}_j \vee \bar{z}_k \vee a_i \vee a_j \vee a_k)$, i.e. $d = 4$.

We now claim that Player I has a winning strategy in the SSG instance (G,s,t) if and only if she has a winning strategy in the corresponding GBF instance (X,C). Note that because Player I wins all the clauses c_k if and only if she wins all the miniclauses c_{kl}, we can argue at the level of the clauses c_k. Note also that Player I wins a clause c_k if and only if she can satisfy any one of the disjuncts $(z_k \equiv \bar{z}_i \vee \bar{z}_j)$, $(z_k \equiv a_k)$, $(z_i \equiv a_i)$, $(z_j \equiv a_j)$. Let us call the first one the "gate term" and the others the "escape terms", with correspondingly the z_k the "gate variables" and the a_k the "escape variables".

Suppose first that Player I has a winning strategy on the SSG instance (G,s,t). She will follow this strategy first on the input variables $x_1,\ldots,x_n$, and will then assign to the gate variables z_k the values that they would have in the correct evaluation of the W circuit on the given input vector x. It is not necessary to assign these values in the topological order of the gates $(g_1,\ldots,g_p)$, but it is of course natural to do so unless the actions of Player II require otherwise.

Since Player I has a winning strategy on the SSG instance, she will also win on the GBF instance unless Player II can make some of the gate variables have different values than what they would have in a correct evaluation of $W(x)$. Let us consider the first time in the play where Player II does something else than sets one of the input variables $x_1,\ldots,x_n$ or assigns a gate variable z_k to its correct value. There are two similar cases to consider:

Case a**:** Player II "cheats" by assigning a gate variable z_k to a value which is either wrong or not yet determined (because the input sequence x has not yet been completely played out). Then Player I sets the corresponding escape variable a_k to the same value, and hence wins all the clauses c where variable z_k appears by default. Player I repeats this response as many times as Player II cheats. Eventually Player II must return to "fair" play (or all the remaining clauses become satisfied by Player I's escape responses), and then also Player I can return to her basic strategy.

Case b**:** Player II assigns one of the escape variables a_k to some value. Player I sets the corresponding gate variable z_k to the same value, and wins all the clauses c where variable z_k appears by default. Play continues as in Case a.

Suppose then that Player I does *not* have a winning strategy in the SSG instance (G,s,t). How could she nevertheless try to win all the clauses c? Let us again consider the first time in the play where Player I does something else than sets one of the input variables $x_1,\ldots,x_n$ or assigns a gate variable z_k to its correct value on a completed input sequence x. There are again two cases to consider:

Case a**:** Player I cheats by assigning a gate variable z_k to a value which is either wrong or not yet determined (because the input sequence x has not yet been completely played out). Then Player II sets the corresponding escape variable a_k to the *opposite* value. This eliminates the escape terms $z_k \equiv a_k$ from all clauses where variable z_k appears, without changing the satisfiability of those clauses otherwise. Hence eventually any inconsistency in the gate terms introduced by Player I will be discovered.

Case b**:** Player I assigns one of the escape variables a_k to some value. If the corresponding gate variable z_k is already assigned, then Player II does nothing, i.e. continues assigning values to the other gate variables in a consistent way. If z_k is still unassigned, then Player II assigns it to the opposite of a_k. Now if $\bar{a}_k$ is the correct value for z_k, then Player I has gained no advantage in the gate variables. If not, then the present Case b reduces to Case a: again the escape terms $z_k \equiv a_k$ have been eliminated from all clauses where variable z_k appears, without changing their satisfiability otherwise, and the newly introduced inconsistency will eventually lead to a gate term which Player I cannot satisfy.

Since the reduction from the given SSG instance to the corresponding GBF instance can be computed in polynomial time, and winning strategies for Player I are preserved, we thus conclude that the problem of deciding GBF-satisfiability is PSPACE-hard.

6 Conclusion and Further Work

We have presented what is to our knowledge the first study of constraint satisfaction games where the order of playing the variables is not restricted. We have established a number of positive results concerning the existence of polynomial-time approximate strategies for unordered constraint satisfaction games for specific constraint types. Also we have proved that GBF – the unordered analogue of QBF – is PSPACE-complete. Some of the pertinent open questions include:

1. Can one improve the given performance bounds on the strategies for even-odd games (Theorem 1) or threshold games (Theorems 2 and 3)? Or can the approximate strategies for threshold games be extended to e.g. bigger monotone constraint families?
2. Can one prove *inapproximability* results for unordered games, similar to those achieved in [9,13] for quantifier-ordered games?
3. What is the smallest value of k such that k-GBF satisfiability is PSPACE-complete? In particular, is 3-GBF satisfiability PSPACE-complete?
4. For which bases is (the analogue of) k-GBF PSPACE-complete? In particular, the even-odd game of Section 3 (2-XOR-GBF) corresponds to the colouring construction game of [5] on two colours.
5. More generally, can one achieve complexity or approximability taxonomies for unordered constraint satisfaction games based on the characteristics of the constraints involved, along the lines of the taxonomies presented in [15] for CSPs or those in [6,7,16] for quantifier-ordered games?

Acknowledgments. This research was supported by the Academy of Finland under grants 128823 (L.A.) and 13136660 (P.O.) Part of the work was done during our visit to the Humboldt-Universität zu Berlin in Autumn 2011, and we thank Sebastian Kuhnert from the Institut für Informatik of HU Berlin for helpful discussions on the topic.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49(5), 672–713 (2002)
2. Ansótegui, C., Gomes, C.P., Selman, B.: The Achilles' heel of QBF. In: Proc. 20th Natl. Conf. on Artificial Intelligence and 17th Conf. on Innovative Applications of Artificial Intelligence (AAAI/IAAI 2005), pp. 275–281 (2005)

3. Benedetti, M., Lallouet, A., Vautard, J.: QCSP made practical by virtue of restricted quantification. In: Proc. 20th Intl. Joint Conf. on Artificial Intelligence (IJCAI 2007), pp. 38–43 (2007)
4. Benedetti, M., Lallouet, A., Vautard, J.: Quantified Constraint Optimization. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 463–477. Springer, Heidelberg (2008)
5. Bodlaender, H.L.: On the complexity of some coloring games. Int. J. Found. Comput. Sci. 2(2), 133–147 (1991)
6. Börner, F., Bulatov, A.A., Chen, H., Jeavons, P., Krokhin, A.A.: The complexity of constraint satisfaction games and QCSP. Inf. Comput. 207(9), 923–944 (2009)
7. Chen, H.: The complexity of quantified constraint satisfaction: Collapsibility, sink algebras, and the three-element case. SIAM J. Comput. 37(5), 1674–1701 (2008)
8. Chen, H., Pál, M.: Optimization, Games, and Quantified Constraint Satisfaction. In: Fiala, J., Koubek, V., Kratochvíl, J. (eds.) MFCS 2004. LNCS, vol. 3153, pp. 239–250. Springer, Heidelberg (2004)
9. Condon, A., Feigenbaum, J., Lund, C., Shor, P.W.: Probabilistically checkable debate systems and nonapproximability of PSPACE-hard functions. Chicago J. Theor. Comput. Sci. 4 (1995)
10. Demaine, E.D.: Playing Games with Algorithms: Algorithmic Combinatorial Game Theory. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 18–32. Springer, Heidelberg (2001)
11. Even, S., Tarjan, R.E.: A combinatorial problem which is complete in polynomial space. J. ACM 23(4), 710–719 (1976)
12. Fraenkel, A.S.: Complexity, appeal and challenges of combinatorial games. Theor. Comput. Sci. 303(3), 393–415 (2004)
13. Hunt III, H.B., Marathe, M.V., Stearns, R.E.: Complexity and approximability of quantified and stochastic constraint satisfaction problems. Electronic Notes in Discrete Mathematics 9, 217–230 (2001)
14. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. Syst. Sci. 9(3), 256–278 (1974)
15. Khanna, S., Sudan, M., Trevisan, L., Williamson, D.P.: The approximability of constraint satisfaction problems. SIAM J. Comput. 30(6), 1863–1920 (2000)
16. Madelaine, F.R., Martin, B.: A tetrachotomy for positive first-order logic without equality. In: Proc. 26th Ann. IEEE Symp. on Logic in Computer Science (LICS 2011), pp. 311–320 (2011)
17. Ramadge, P., Wonham, W.: The control of discrete event systems. Proc. of the IEEE 77(1), 81–98 (1989)
18. Schaefer, T.J.: On the complexity of some two-person perfect-information games. J. Comput. Syst. Sci. 16(2), 185–225 (1978)
19. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time: Preliminary report. In: Proc. 5th Ann. ACM Symp. on Theory of Computing (STOC 1973), pp. 1–9 (1973)

A Polynomial-Time Algorithm for Computing the Maximum Common Subgraph of Outerplanar Graphs of Bounded Degree

Tatsuya Akutsu and Takeyuki Tamura

Bioinformatics Center, Institute for Chemical Research, Kyoto University,
Gokasho, Uji, Kyoto 611-0011, Japan
{takutsu,tamura}@kuicr.kyoto-u.ac.jp

Abstract. This paper considers the maximum common subgraph problem, which is to find a connected graph with the maximum number of edges that is isomorphic to a subgraph of each of the two input graphs. This paper presents a dynamic programming algorithm for computing the maximum common subgraph of two outerplanar graphs whose maximum vertex degree is bounded by a constant, where it is known that the problem is NP-hard even for outerplanar graphs of unbounded degree. Although the algorithm repeatedly modifies input graphs, it is shown that the number of relevant subproblems is polynomially bounded and thus the algorithm works in polynomial time.

Keywords: maximum common subgraph, outerplanar graph, dynamic programming.

1 Introduction

Comparison of graph-structured data is important and fundamental in computer science. Among many graph comparison problems, the *maximum common subgraph problem* has applications in various areas, which include pattern recognition [4,14] and chemistry [12]. Although there exist several variants, the maximum common subgraph problem (MCS) usually means the problem of finding a connected graph with the maximum number of edges that is isomorphic to a subgraph of each of the two input undirected graphs.

Due to its importance in pattern recognition and chemistry, many practical algorithms have been developed for MCS and its variants [4,12,14]. Some exponential-time algorithms better than naive ones have also been developed [1,9]. Kann studied the approximability of MCS and related problems [10].

It is also important for MCS to study polynomially solvable subclasses of graphs. It is well-known that if input graphs are trees, MCS can be solved in polynomial time using maximum weight bipartite matching [6]. Akutsu showed that MCS can be solved in polynomial time if input graphs are almost trees of bounded degree whereas MCS remains NP-hard for almost trees of unbounded degree [2], where a graph is called almost tree if it is connected and the number

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 76–87, 2012.

of edges in each biconnected component is bounded by the number of vertices plus some constant. Yamaguchi et al. developed a polynomial-time algorithm for MCS and the maximum common induced connected subgraph problem for a degree bounded partial k-tree and a graph with a polynomially bounded number of spanning trees, where k is a constant [16]. However, the latter condition seems too strong. Schietgat et al. developed a polynomial-time algorithm for outerplanar graphs under the block-and-bridge preserving subgraph isomorphism [13]. However, they modified the definition of MCS by this restriction. Although it was announced that MCS can be solved in polynomial time if input graphs are partial k-trees and MCS must be k-connected (for example, see [3]), the restriction that subgraphs are k-connected is too strict from a practical viewpoint. On the subgraph isomorphism problem, which is closely related to MCS, polynomial-time algorithms have been developed for biconnected outerplanar graphs [11,15] and for partial k-trees with some constraints as well as their extensions [5,7].

In this paper, we present a polynomial-time algorithm for outerplanar graphs of bounded degree. Although this graph class is not a superset of the classes in previous studies [2,16], it covers a wide range of chemical compounds[1]. Furthermore, the algorithm or its analysis in this paper is not a simple extension or variant of that for the subgraph isomorphism problem for outerplanar graphs [11,15] or partial k-trees [5,7]. These algorithms heavily depend on the property that each connected component in a subgraph is not decomposed. However, to be discussed in Section 4, connected components from both input graphs can be decomposed in MCS and considering all decompositions easily leads to exponential-time algorithms. In order to cope with this difficulty, we introduce the concept of *blade*. The blade and its analysis play a key role in this paper.

2 Preliminaries

A graph is called *outerplanar* if it can be drawn on a plane so that all vertices lie on the outer face (i.e., the unbounded exterior region) without crossing of edges. Although there exist many embeddings (i.e., drawings on a plane) of an outerplanar graph, it is known that one embedding can be computed in linear time. Therefore, we assume in this paper that each graph is given with its planar embedding. A path is called *simple* if it does not pass the same vertex multiple times. In this paper, a path always means a simple path that is not a cycle.

A *cutvertex* of a connected graph is a vertex whose removal disconnects the graph. A graph is *biconnected* if it is connected and does not have a cutvertex. A maximal biconnected subgraph is called a *biconnected component*. A biconnected component is called a *block* if it consists of at least three vertices, otherwise it is an edge and called a *bridge*. An edge in a block is called an *outer* edge if it lies on the boundary of the outer face, otherwise called an *inner* edge. It is well-known that any block of an outerplanar graph has a unique Hamiltonian cycle, which consists of outer edges only.

[1] It was reported that 94.4% of chemical compounds in NCI database have outerplanar graph structures [8].

If we fix an arbitrary vertex of a graph G as the root r, we can define the parent-child relationship on biconnected components. For two vertices u and v, u is called *further* than v if every simple path from u to r contains v. A biconnected component C is called a *parent* of a biconnected component C' if C and C' share a vertex v, where v is uniquely determined, and every path from any vertex in C' to the root contains v. In such a case, C' is called a *child* of C. A cutvertex v is also called a *parent* of C if v is contained in both C and its parent component[2]. Furthermore, the root r is a parent of C if r is contained in C.

For each cutvertex v, $G(v)$ denotes the subgraph of G induced by v and the vertices further than v. For a pair of a cutvertex v and a biconnected component C containing v, $G(v, C)$ denotes the subgraph of G induced by vertices in C and its descendant components. For a biconnected component B with its parent cutvertex w, a pair of vertices v and v' in B is called a *cut pair* if $v \neq v'$, $v \neq w$, and $v' \neq w$ hold. For a pair (v, v') in B such that $v \neq v'$ holds (v or v' can be the parent cutvertex), $VB(v, v')$ denotes the set of the vertices lying on the one of the two paths connecting v and v' in the Hamilton cycle that does not contain the parent cutvertex except its endpoints. $B(v, v')$ is the subgraph of B induced by $VB(v, v')$ and is called a *half block*. It is to be noted that $B(v, v')$ contains both v and v'. Then, $G(v, v')$ denotes the subgraph of G induced by $VB(v, v')$ and the vertices in the biconnected components each of which is a descendant of some vertex in $VB(v, v') - \{v, v'\}$, and $\overline{G}(v, v')$ denotes the subgraph of G induced by the vertices in $G(v, v')$ and descendant components of v and v'.

Example. Fig. 1 shows an example of an outerplanar graph $G(V, E)$. Blocks and bridges are shown by gray regions and bold lines, respectively. B_1, B_3 and e_2 are the children of the root r. B_4, B_6 and B_7 are the children of B_3, whereas B_4 and B_6 are the children of w. Both w and B_3 are the parents of B_4 and B_6. $G(w)$ consists of B_4, B_5 and B_6, whereas $G(w, B_4)$ consists of B_4 and B_5. (v, v') is a cut pair of B_7, and $B_7(v, v')$ is a region surrounded by a dashed bold curve. $\overline{G}(v, v')$ consists of $B_7(v, v')$, B_8, B_9, B_{10}, e_4, e_5, and e_6, whereas $G(v, v')$ consists of $B_7(v, v')$, B_{10}, e_4, and e_5.

If a connected graph $G_c(V_c, E_c)$ is isomorphic to a subgraph of G_1 and a subgraph of G_2, we call G_c a *common subgraph* of G_1 and G_2. A common subgraph G_c is called a *maximum common subgraph* (MCS) of G_1 and G_2 if its number of edges is the maximum among all common subgraphs[3]. In this paper, we consider the following problem.

Maximum Common Subgraph of Outerplanar Graphs of Bounded Degree (OUTER-MCS)
Given two undirected connected outerplanar graphs G_1 and G_2 whose maximum vertex degree is bounded by a constant D, find a maximum common subgraph of G_1 and G_2.

[2] Both of a cutvertex and a biconnected component can be parents of the same component.

[3] We use MCS to denote both the problem and the maximum common subgraph.

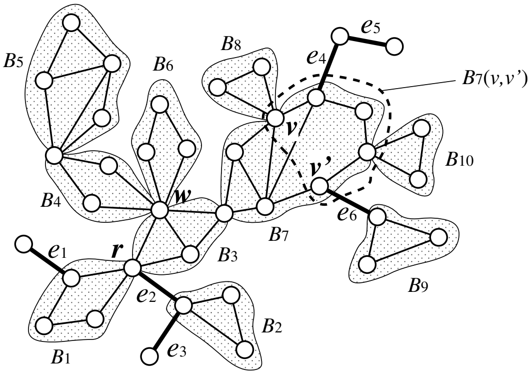


Fig. 1. Example of an outerplanar graph

Notice that the degree bound is essential because MCS is NP-hard for outerplanar graphs of unbounded degree even if each biconnected component consists of at most three vertices [2]. Although we do not consider labels on vertices or edges, our results can be extended to vertex-labeled and/or edge-labeled cases in which label information must be preserved in isomorphic mapping. In the following, n denotes the maximum number of vertices of two input graphs[4].

In this paper, we implicitly make extensive use of the following well-known fact [11] along with outerplanarity of input graphs.

Fact 1. *Let G_1 and G_2 be biconnected outerplanar graphs. Let $(u_1, u_2, \ldots, u_m)$ (resp. $(v_1, v_2, \ldots, v_n)$) be the vertices of G_1 (resp. G_2) arranged in the clockwise order in some planar embedding of G_1 (resp. G_2). If there is an isomorphic mapping $\{(u_1, v_{i_1}), (u_2, v_{i_2}), \ldots, (u_m, v_{i_m})\}$ from G_1 to a subgraph of G_2 then $v_{i_1}, v_{i_2}, \ldots, v_{i_m}$ appear in G_2 in either clockwise or counterclockwise order.*

3 Algorithm for a Restricted Case

In this section, we consider the following restricted variant of OUTER-MCS, which is called SIMPLE-OUTER-MCS, and present a polynomial-time algorithm for it: (i) any two vertices in different biconnected components in a maximum common subgraph G_c must not be mapped to vertices in the same biconnected component in G_1 (resp. G_2), (ii) each bridge in G_c must be mapped to a bridge in G_1 (resp. G_2), (iii) the maximum degree need not be bounded.

It is to be noted from the definition of a common subgraph (regardless of the above restrictions) that no two vertices in different biconnected components in G_1 (resp. G_2) are mapped to vertices in the same biconnected component in any common subgraph, or no bridge in G_1 (resp. G_2) is mapped to an edge in a block in any common subgraph.

[4] It should be noted that the number of vertices and the number of edges are in the same order since we only consider connected outerplanar graphs.

It seems that SIMPLE-OUTER-MCS is the same as one studied by Schietgat et al. [13]. Although our algorithm is more complex and less efficient than theirs, we present it here because the algorithm for a general (but bounded degree) case is rather involved and is based on our algorithm for SIMPLE-OUTER-MCS.

Here we present a recursive algorithm to compute the size of MCS in SIMPLE-OUTER-MCS, which can be easily transformed into a dynamic programming algorithm to compute an MCS. The following is the main procedure of the recursive algorithm.

```
Procedure $SimpleOuterMCS(G_1, G_2)$
$s_{\max} \leftarrow 0$;
**for all** pairs of vertices $(u, v) \in V_1 \times V_2$ **do**
  Let $(u, v)$ be the root pair $(r_1, r_2)$ of $(G_1, G_2)$;
  $s_{\max} \leftarrow \max(s_{\max}, MCS_c(G_1(r_1), G_2(r_2)))$;
**return** $s_{\max}$.
```

The algorithm consists of recursive computation of the following three scores:

$MCS_c(G_1(u), G_2(v))$**:** the size of an MCS G_c between $G_1(u)$ and $G_2(v)$, where (u, v) is a pair of the roots or a pair of cutvertices, and G_c must contain a vertex corresponding to both u and v.

$MCS_b(G_1(u, C), G_2(v, D))$**:** the size of an MCS G_c between $G_1(u, C)$ and $G_2(v, D)$, where (C, D) is either a pair of blocks or a pair of bridges, u (resp. v) is the cutvertex belonging to both C (resp. D) and its parent, G_c must contain a vertex corresponding to both u and v, and G_c must contain a biconnected component (which can be empty) corresponding to a subgraph of C and a subgraph D.

$MCS_p(G_1(u, u'), G_2(v, v'))$**:** the size of an MCS G_c between $G_1(u, u')$ and $G_2(v, v')$, where (u, u') (resp. (v, v')) is a cut pair, and G_c must contain a cut pair (w, w') corresponding to both (u, u') and (v, v'). If there does not exist such G_c (which must be connected), its score is $-\infty$.

In the following, we describe how to compute these scores.

Computation of $MCS_c(G_1(u), G_2(v))$

As in the dynamic programming algorithm for MCS for trees or almost trees [2], we construct a bipartite graph and compute a maximum weight matching.

Let $C_1, \ldots, C_{h_1}, e_1, \ldots, e_{h_2}$ and $D_1, \ldots, D_{k_1}, f_1, \ldots, f_{k_2}$ be children of u and v respectively, where C_is and D_js are blocks and e_is and f_js are bridges (see Fig. 2). We construct an edge-weighted bipartite graph $BG(X, Y; E)$ by

$$\begin{aligned} X = \{C_1, \ldots, C_{h_1}, e_1, \ldots, e_{h_2}\}, \quad Y &= \{D_1, \ldots, D_{k_1}, f_1, \ldots, f_{k_2}\}, \\ E = \{(x, y) \mid x \in X, y \in Y\}, & \\ w(C_i, D_j) = MCS_b(G_1(u, C_i), G_2(v, D_j)), \quad w(C_i, f_j) &= 0, \\ w(e_i, f_j) = MCS_b(G_1(u, e_i), G_2(v, f_j)), \quad w(e_i, D_j) &= 0. \end{aligned}$$

Then, we let $MCS_c(G_1(u), G_2(v))$ be the weight of the maximum weight bipartite matching of $BG(X, Y; E)$.

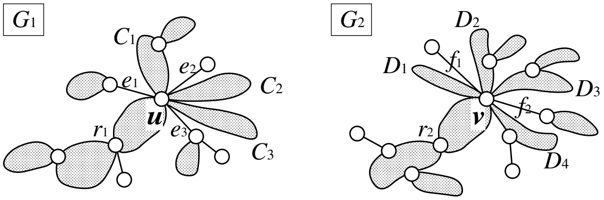


Fig. 2. Computation of $MCS_c(G_1(u), G_2(v))$

Computation of $MCS_b(G_1(u, C), G_2(v, D))$

Let $(u_1, u_2, \ldots, u_h)$ be the sequence of vertices in $G_1(u, C)$ such that there exists an edge $\{u_i, u\}$ for each u_i, where $u_1, u_2, \ldots, u_h$ are arranged in the clockwise order. $(v_1, v_2, \ldots, v_k)$ is defined for $G_2(v, D)$ in the same way. A pair of subsequences $((u_{i_1}, u_{i_2}, \ldots, u_{i_g}), (v_{j_1}, v_{j_2}, \ldots, v_{j_g}))$ is called an *alignment* if $i_1 < i_2 < \cdots < i_g$, and $j_1 < j_2 < \cdots < j_g$ or $j_g < j_{g-1} < \cdots < j_1$ hold[5] where $g = 0$ is allowed. We compute $MCS_b(G_1(u, C), G_2(v, D))$ by the following (see Fig. 3).

```
Procedure $SimpleOuterMCS_b(G_1(u, C), G_2(v, D))$
$s_{\max} \leftarrow 0$;
**for all** alignments $((u_{i_1}, u_{i_2}, \ldots, u_{i_g}), (v_{j_1}, v_{j_2}, \ldots, v_{j_g}))$ **do**;
   **if** $C$ is a block and $g = 1$ **then continue**;  /* blocks must be preserved */
   $s \leftarrow 0$;
   **for** $t = 1$ **to** $g$ **do** $s \leftarrow s + 1 + MCS_c(G_1(u_{i_t}), G_2(v_{j_t}))$;
   **for** $t = 2$ **to** $g$ **do** $s \leftarrow s + MCS_p(G_1(u_{i_{t-1}}, u_{i_t}), G_2(v_{j_{t-1}}, v_{j_t}))$;
   $s_{\max} \leftarrow \max(s, s_{\max})$;
**return** $s_{\max}$.
```

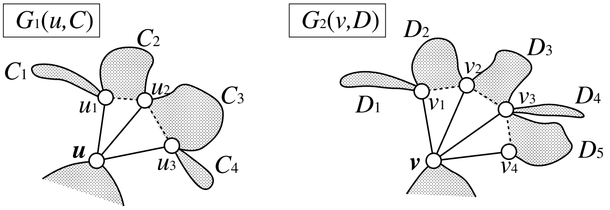


Fig. 3. Computation of $MCS_b(G_1(u, C), G_2(v, D))$

For example, consider an alignment $((u_1, u_2, u_3), (v_1, v_2, v_4))$ in Fig. 3, where all alignments are to be examined in the algorithm. Then, the score of this alignment is given by $3 + MCS_b(G_1(u_1, C_1), G_2(v_1, D_1)) + MCS_p(G_1(u_1, u_2),$

[5] The latter ordering is required for handling mirror images.

$G_2(v_1, v_2))$ $+ MCS_p(G_1(u_2, u_3), G_2(v_2, v_4))$. In this case, an edge $\{v, v_3\}$ is removed and then v_3 is treated as a vertex on the path connecting v_2 and v_4 in the outer face.

Since the above procedure examines all possible alignments, it may take exponential time. However, we can modify it into a dynamic programming procedure as shown below, where we omit a subprocedure for handling mirror images. In this procedure, $u_1, u_2, \ldots, u_h$ and $v_1, v_2, \ldots, v_k$ are processed from left to right. In the first **for** loop, $M[s, t]$ stores the size of MCS between $G_1(u_s)$ and $G_2(v_t)$ plus one (corresponding to a common edge between $\{u, u_s\}$ and $\{v, v_t\}$). The second double **for** loop computes an optimal alignment. $M[s, t]$ stores the size of MCS between $G_1(u, C)$ and $G_2(v, D)$ up to u_s and v_t, respectively. $flag$ is introduced to ensure the connectedness of a common subgraph. For example, $flag = 0$ if $G_1(u)$ is a triangle but $G_2(v)$ is a rectangle.

```
for all $(s,t) \in \{1,\ldots,h\} \times \{1,\ldots,k\}$ do
  $M[s,t] \leftarrow 1 + MCS_c(G_1(u_s), G_2(v_t))$;
$flag \leftarrow 0$;
for $s = 2$ to $h$ do
  for $t = 2$ to $k$ do
    $M[s,t] \leftarrow M[s,t]+$
                $\max_{s'<s,t'<t}\{M[s',t'] + MCS_p(G_1(u_{s'}, u_s), G_2(u_{t'}, u_t))\}$;
    if $M[s,t] > -\infty$ then $flag \leftarrow 1$;
if $C$ is a block and $flag = 0$ then return 0 else return $\max_{s,t} M[s,t]$.
```

Computation of $MCS_p(G_1(u, u'), G_2(v, v'))$

Let $(u_1, u_2, \ldots, u_h)$ be the sequence of vertices in $G_1(u, u')$ such that there exists an edge $\{u_i, u\}$ or $\{u_i, u'\}$ for each u_i, where $u_1, u_2, \ldots, u_h$ are arranged in the clockwise order. $(v_1, v_2, \ldots, v_k)$ is defined for $G_2(v, v')$ in the same way. For a pair (u_i, v_j), $l(u_i, v_j) = 1$ if $\{u_i, u\} \in E_1$ and $\{v_j, v\} \in E_2$ hold, otherwise $l(u_i, v_j) = 0$. For a pair (u_i, v_j), $r(u_i, v_j) = 1$ if $\{u_i, u'\} \in E_1$ and $\{v_j, v'\} \in E_2$ hold, otherwise $r(u_i, v_j) = 0$. We compute $MCS_p(G_1(u, u'), G_2(v, v'))$ by the following procedure, where it does not examine alignments with $j_g < j_{g-1} < \cdots < j_1$.

```
Procedure $SimpleOuterMCS_p(G_1(u,u'), G_2(v,v'))$
if $\{u,u'\} \in E_1$ and $\{v,v'\} \in E_2$ then $s_{\max} \leftarrow 1$ else $s_{\max} \leftarrow -\infty$;
for all alignments $((u_{i_1}, u_{i_2}, \ldots, u_{i_g}), (v_{j_1}, v_{j_2}, \ldots, v_{j_g}))$ do
  if $l(u_{i_t}, v_{j_t}) = 0$ and $r(u_{i_t}, v_{j_t}) = 0$ hold for some $t$ then continue;
  if $l(u_{i_1}, v_{j_1}) = 0$ and $r(u_{i_g}, v_{j_g}) = 0$ hold then continue;
  if $\{u,u'\} \in E_1$ and $\{v,v'\} \in E_2$ then $s \leftarrow 1$ else $s \leftarrow 0$;
  for $t = 1$ to $g$ do $s \leftarrow s + l(u_{i_t}, v_{j_t}) + r(u_{i_t}, v_{j_t}) + MCS_c(G_1(u_{i_t}), G_2(v_{j_t}))$;
  for $t = 2$ to $g$ do $s \leftarrow s + MCS_p(G_1(u_{i_{t-1}}, u_{i_t}), G_2(v_{j_{t-1}}, v_{j_t}))$;
  $s_{\max} \leftarrow \max(s, s_{\max})$;
return $s_{\max}$.
```

This procedure returns $-\infty$ if there does not exist a connected common subgraph between $G_1(u, u')$ and $G_2(v, v')$ that contains (w, w') corresponding to both (u, u') and (v, v').

As an example, consider an alignment $((u_1,u_2,u_3,u_4),(v_1,v_2,v_3,v_5))$ in Fig. 4. Then, the score is given by $4+MCS_p(G_1(u_1,u_2),G_2(v_1,v_2))+MCS_p(G_1(u_2,u_3), G_2(v_2,v_3))+MCS_b(G_1(u_3,C_3),G_2(v_3,D_4))+MCS_p(G_1(u_3,u_4),G_2(v_3.v_5))$. For another example, the score is $-\infty$ for each of alignments $((u_1,u_3),(v_4,v_5))$, $((u_1,u_2),(v_1,v_2))$, and $((u_3),(v_3))$, whereas the score of $((u_2),(v_3))$ is 2.

As in the case of $SimpleOuterMCS_b(G_1(u,C),G_2(v,D))$, $SimpleOuter$ MCS_p $(G_1(u,u'),G_2(v,v'))$ can be modified into a dynamic programming version.

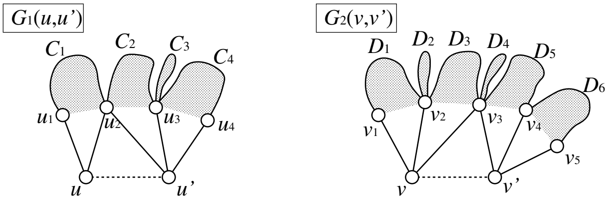


Fig. 4. Computation of $MCS_p(G_1(u,u'),G_2(v,v'))$

Then, we have the following theorem, where the proof is omitted here.

Theorem 1. *SIMPLE-OUTER-MCS can be solved in polynomial time.*

4 Algorithm for Outerplanar Graphs of Bounded Degree

In order to extend the algorithm in Section 3 for a general (but bounded degree) case, we need to consider decomposition of biconnected components. For example, consider graphs G_1 and G_2 in Fig. 5. We can see that in order to obtain a maximum common subgraph, biconnected components in G_1 and G_2 should be decomposed as shown in Fig. 5, where there are several other ways of optimal decompositions. This is the crucial point because considering all possible decompositions easily leads to exponential-time algorithms. In order to characterize decomposed components, we introduce the concept of *blade* as below.

Suppose that $v_{i_1},\ldots,v_{i_k}$ are the vertices of a half block arranged in this order, and v_{i_1} and v_{i_k} are respectively connected to v and v', where v and v' can be

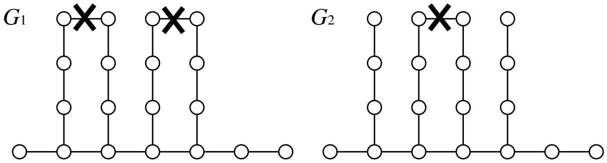


Fig. 5. Example of a difficult case

the same vertex. If we cut one edge $\{v_{i_h}, v_{i_{h+1}}\}$, we obtain two subgraphs, one induced by $v_{i_1}, v_{i_2}, \ldots, v_{i_h}$ and the other induced by $v_{i_k}, v_{i_{k-1}}, \ldots, v_{i_{h+1}}$, where only one such subgraph is obtained in the case of $i_1 = i_h$ or $i_k = i_{h+1}$, and no such subgraph is obtained in the case of $k = 2$. Each of these components is a chain of biconnected components called a *blade body*, and a subgraph consisting of a blade body and its descendants is called a *blade* (see Fig. 6). Vertices v_{i_1} and v_{i_k}, an edge $\{v_{i_h}, v_{i_{h+1}}\}$, and vertices $v_{i_h}, v_{i_{h+1}}$ are called *base vertices*, a *tip edge*, and *tip vertices*, respectively. The sequence of edges in the shortest path from v_{i_1} to v_{i_h} (resp. from v_{i_k} to $v_{i_{h+1}}$) is called the *backbone* of a blade. If $\{v, v_{i_1}\}$ is the leftmost edge (resp. $\{v', v_{i_k}\}$ is the rightmost edge) connecting to v (resp. v') and is removed, the resulting half block induced by $v_{i_k}, \ldots, v_{i_2}, v_{i_1}$ (resp. $(v_{i_1}, v_{i_2}, \ldots, v_{i_k})$) is also regarded as a blade body where v_{i_k} (resp. v_{i_1}) becomes the base vertex. For example, the rightmost blade in Fig. 7 is created by removing the rightmost edge of C_1.

Since a blade can be specified by a pair of base and tip vertices and an orientation (clockwise or counterclockwise), there exist $O(n^2)$ blades in G_1 and G_2. Of course, we need to consider the possibility that during the execution of the algorithm, other subgraphs may appear from which new blades are created. However, we will show later that blades appearing in the algorithm are restricted to be those in G_1 and G_2.

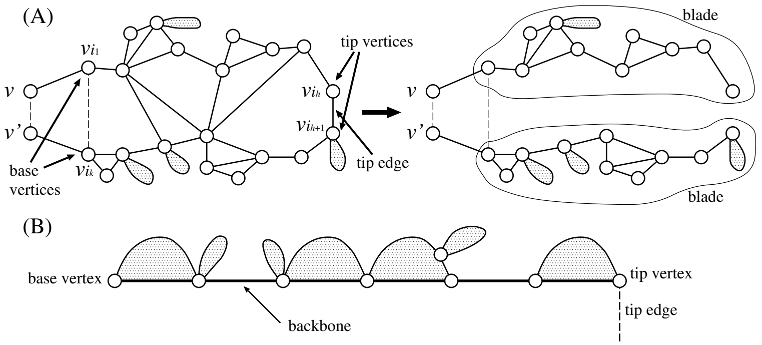


Fig. 6. (A) Construction of blades where subgraphs excluding gray regions (descendant components) are blade bodies, and (B) schematic illustration of a blade

4.1 Description of Algorithm

In this subsection, we describe the algorithm as a recursive procedure, which can be transformed into a dynamic programming one as in Section 3.

The main procedure ($OuterMCS(G_1, G_2)$) is the same as in Section 3, and we recursively compute three kinds of scores: $MCS_c(G_1(u), G_2(v))$, $MCS_b(G_1(u, C)$, $G_2(v, D))$, and $MCS_p(G_1(u, u'), G_2(v, v'))$, where cutvertices, cut pairs, blocks, and bridges do not necessarily mean those in the original graphs but may mean those in subgraphs generated by the algorithm.

Computation of $MCS_c(G_1(u), G_2(v))$

Let $C_1, \ldots, C_{h_1}$ and $e_1, \ldots, e_{h_2}$ be children of u, where C_is and e_js are blocks and bridges, respectively. Let $u_{i_1}, \ldots, u_{i_h}$ be the neighboring vertices of u that are contained in children of u. We define a *configuration* as a tuple of the following (see Fig. 7).

$s(u_{i_j}) \in \{0, 1\}$ for $j = 1, \ldots, k$: $s(u_{i_j}) = 1$ means that u_{i_j} is selected as a neighbor of u in a common subgraph, otherwise $s(u_{i_j}) = 0$. u_{i_j} is called a *selected vertex* if $s(u_{i_j}) = 1$.

$tip(u_{i_j}, u_{i_k})$: $e = tip(u_{i_j}, u_{i_k})$ is an edge in $B(u_{i_j}, u_{i_k})$ where B is the block containing u_{i_j}, u_{i_k}, and u. This edge is defined only for a consecutive selected vertex pair u_{i_j} and u_{i_k} in the same block (i.e., $B(u_{i_j}, u_{i_k})$ does not contain any other selected vertex). e is used as a tip edge where e can be empty which means that we do not cut any edge in $B(u_{i_j}, u_{i_k})$. It is to be noted that at most one edge in $B(u_{i_j}, u_{i_k})$ can be a tip edge and thus each $B(u_{i_j}, u_{i_k})$ is divided into at most two blade bodies: further decomposition will be done in later steps.

Each configuration defines a subgraph of $G_1(u)$ as follows.

- $e_i = \{u_{i_j}, u\}$ ($i \in \{1, \ldots, h_2\}$) remains if $s(u_{i_j}) = 1$. Otherwise e_i is removed along with its descendants.
- If no vertex in C_i is selected, C_i is removed along with its descendants. Otherwise, half blocks in C_i are broken into blade bodies (according to $s(\ldots)$s and $tip(\ldots)$s) and edges $\{u_{i_j}, u\}$ with $s(u_{i_j}) = 0$ are removed.

Let $C'_1, \ldots, C'_{p_1}$ and $e'_1, \ldots, e'_{p_2}$ be the resulting blocks and bridges containing u, which are new 'children' of u, for a configuration F_1. Configurations are defined for $G_2(v)$ in an analogous way. Let $D'_1, \ldots, D'_{q_1}$ and $f'_1, \ldots, f'_{q_2}$ be the resulting new children of v for a configuration F_2 of G_2. As in Section 3, we construct a bipartite graph BG_{F_1,F_2} by $w(C'_i, D'_j) = MCS_b(G_1(u, C'_i), G_2(v, D'_j))$, $w(C'_i, f'_j) = 0$, $w(e'_i, f'_j) = MCS_b(G_1(u, e'_i), G_2(v, f'_j))$, $w(e'_i, D'_j) = 0$, and compute the weight of the maximum weight matching for each configuration pair (F_1, F_2)[6]. The following is a procedure for computing $MCS_c(G_1(u), G_2(v))$.

```
Procedure $OuterMCS_c(G_1(u), G_2(v))$
$s_{\max} \leftarrow 0$;
for all configurations $F_1$ for $G_1(u)$ do
   for all configurations $F_2$ for $G_2(v)$ do
      $s \leftarrow$ weight of the maximum weight matching of $BG_{F_1,F_2}$;
      if $s > s_{\max}$ then $s_{\max} \leftarrow s$;
return $s_{\max}$.
```

[6] Although a bridge cannot be mapped on a block here, a bridge can be mapped to an edge in a block by cutting the block using tip edge(s).

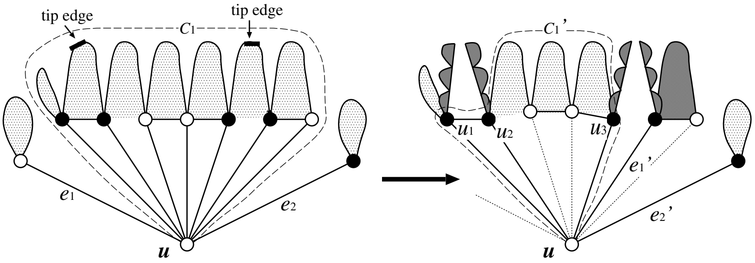


Fig. 7. Example of configuration and its resulting subgraph of $G_1(u)$, where black circles, dark gray regions, thin dotted lines denote selected vertices, blades, and removed edges, respectively. C_1' has three blades and one block as the children, and e_1' has two blades as the children. The role of u_1, u_2, and u_3 corresponds to that of u_1, u_2, and u_3 in Fig. 3.

Computation of $MCS_b(G_1(u, C'), G_2(v, D'))$

This score can be computed as in Section 3. In this case, we can directly examine all possible alignments because the number of neighbors of u or v is bounded by a constant and we need to examine a constant number of alignments.

Computation of $MCS_p(G_1(u, u'), G_2(v, v'))$**.**

This part is a bit more complex than the restricted case because we need to take configurations into account, where the details are omitted here.

4.2 Analysis

It is straightforward to check the correctness of the algorithm because it implicitly examines all possible common subgraphs. Therefore, we focus on analysis of the time complexity, where the proofs are omitted here. As mentioned before, each blade is specified by base and tip vertices in G_1 or G_2 and an orientation. Each half block is also specified by two vertices in a block in G_1 or G_2. We show that this property is maintained throughout the execution of the algorithm and bound the number of half blocks and blades as below.

Lemma 1. *The number of different half blocks and blades appearing in* $OuterMCS(G_1, G_2)$ *is* $O(n^2)$*.*

Finally, we obtain the following theorem.

Theorem 2. *A maximum connected common subgraph of two outerplanar graphs of bounded degree can be computed in polynomial time.*

5 Concluding Remarks

We have presented a polynomial-time algorithm for the maximum common subgraph problem for outerplanar graphs of bounded degree. However, it is not practically efficient. Therefore, development of a much faster algorithm is left as an open problem. Although the proposed algorithm might be modified for outputting all maximum common subgraphs, it would not be an output-polynomial time algorithm. Therefore, such an algorithm should also be developed.

References

1. Abu-Khzam, F.N., Samatova, N.F., Rizk, M.A., Langston, M.A.: The maximum common subgraph problem: faster solutions via vertex cover. In: Proc. 2007 IEEE/ACS Int. Conf. Computer Systems and Applications, pp. 367–373. IEEE (2007)
2. Akutsu, T.: A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. IEICE Trans. Fundamentals E76-A, 1488–1493 (1993)
3. Bachl, S., Brandenburg, F.-J., Gmach, D.: Computing and drawing isomorphic subgraphs. J. Graph Algorithms and Applications 8, 215–238 (2004)
4. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. Int. J. Pattern Recognition and Artificial Intelligence 18, 265–298 (2004)
5. Dessmark, A., Lingas, A., Proskurowski, A.: Faster algorithms for subgraph isomorphism of k-connected partial k-trees. Algorithmica 27, 337–347 (2000)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, New York (1979)
7. Hajiaghayi, M., Nishimura, N.: Subgraph isomorphism, log-bounded fragmentation and graphs of (locally) bounded treewidth. J. Comput. Syst. Sci. 73, 755–768 (2007)
8. Horváth, T., Ramon, J., Wrobel, S.: Frequent subgraph mining in outerplanar graphs. In: Proc. 12th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, pp. 197–206. ACM (2006)
9. Huang, X., Lai, J., Jennings, S.F.: Maximum common subgraph: some upper bound and lower bound results. BMC Bioinformatics 7(suppl. 4), S-4 (2006)
10. Kann, V.: On the Approximability of the Maximum Common Subgraph Problem. In: Finkel, A., Jantzen, M. (eds.) STACS 1992. LNCS, vol. 577, pp. 377–388. Springer, Heidelberg (1992)
11. Lingas, A.: Subgraph isomorphism for biconnected outerplanar graphs in cubic time. Theoret. Comput. Sci. 63, 295–302 (1989)
12. Raymond, J.W., Willett, P.: Maximum common subgraph isomorphism algorithms for the matching of chemical structures. J. Computer-Aided Molecular Design 16, 521–533 (2002)
13. Schietgat, L., Ramon, J., Bruynooghe, M.: A polynomial-time metric for outerplanar graphs. In: Proc. Workshop on Mining and Learning with Graphs (2007)
14. Shearer, K., Bunke, H., Venkatesh, S.: Video indexing and similarity retrieval by largest common subgraph detection using decision trees. Pattern Recognition 34, 1075–1091 (2001)
15. Syslo, M.M.: The subgraph isomorphism problem for outerplanar graphs. Theoret. Comput. Sci. 17, 91–97 (1982)
16. Yamaguchi, A., Aoki, K.F., Mamitsuka, H.: Finding the maximum common subgraph of a partial k-tree and a graph with a polynomially bounded number of spanning trees. Inf. Proc. Lett. 92, 57–63 (2004)

Reductions to the Set of Random Strings: The Resource-Bounded Case

Eric Allender[1], Harry Buhrman[2], Luke Friedman[1], and Bruno Loff[3]

[1] Department of Computer Science, Rutgers University, Piscataway, NJ 08855, USA
{allender,lbfried}@cs.rutgers.edu
[2] CWI and University of Amsterdam
buhrman@cwi.nl
[3] CWI
bruno.loff@gmail.com

Abstract. This paper is motivated by a conjecture [1,5] that BPP can be characterized in terms of polynomial-time nonadaptive reductions to the set of Kolmogorov-random strings. In this paper we show that an approach laid out in [5] to settle this conjecture cannot succeed without significant alteration, but that it does bear fruit if we consider time-bounded Kolmogorov complexity instead.

We show that if a set A is reducible in polynomial time to the set of time-t-bounded Kolmogorov-random strings (for all large enough time bounds t), then A is in P/poly, and that if in addition such a reduction exists for any universal Turing machine one uses in the definition of Kolmogorov complexity, then A is in PSPACE.

1 Introduction

The roots of this investigation stretch back to the discovery that PSPACE $\subseteq$ P^R and NEXP $\subseteq$ NP^R, where R is the set of Kolmogorov-random strings [4,3]. Later, it was shown that BPP $\subseteq$ P$^R_{tt}$ [8], where P$^A_{tt}$ denotes the class of problems reducible to A via polynomial-time *nonadaptive* (or *truth-table*) reductions.

There is evidence indicating that some of these inclusions are in some sense optimal. The inclusions mentioned in the preceding paragraph hold for the two most-common versions of Kolmogorov complexity (the plain complexity C and the prefix-free complexity K), and they also hold no matter which universal Turing machine one uses when defining the measures K and C.

Let R_{K_U} denote the set of random strings according to the prefix-free measure K given by the universal machine U: $R_{K_U} = \{x : K_U(x) \geq |x|\}$. Last year, it was shown that the class of decidable sets that are polynomial-time truth-table reducible to R_{K_U} for every U is contained in PSPACE [6]. That is, although P$^{R_{K_U}}_{tt}$ contains arbitrarily complex decidable sets, an extremely complex set can only be there because of characteristics of R_{K_U} that are fragile with respect to the choice of U.

This motivates the following definition: DTTR is the class of all decidable problems that are polynomial-time truth-table reducible to R_{K_U} for every choice of universal prefix-free Turing machine U. Thus it was proven that BPP $\subseteq$ DTTR $\subseteq$ PSPACE $\subseteq$ P^{R_K}, which leads naturally to the following:

Research question: *Does* DTTR *sit closer to* BPP*, or closer to* PSPACE*?*

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 88–99, 2012.

A conjecture by various authors [5,1] is that DTTR actually characterizes BPP exactly. Part of the intuition is that a non-adaptive reduction cannot make use of queries to R_K larger than $O(\log n)$ to solve a decidable problem. If indeed true we could use the strings of length at most $O(\log n)$ as advice and answer the larger queries with NO, to show that these sets are in P/poly. The rest of the intuition is that the smaller strings can only be used as a source for pseudo-randomness. If we are able to prove this conjecture, then we can make use of the tools of Kolmogorov complexity to study various questions about the class BPP. Because of the inclusions listed above, this now amounts to understanding the relative power of Turing reductions *vs.* truth-table reductions to R_K.

In an attempt to tackle this question, it was conjectured in [5,1] that the DTTR $\subseteq$ PSPACE upper bound can be improved to PSPACE $\cap$ P/poly, and an approach was suggested, based on the above mentioned intuition in connection with formal systems of arithmetic. In this paper, we show that this approach must fail, or at least requires significant changes. Interestingly, we can also prove that this intuition — that the large queries can be answered with NO — *can* be used in the resource-bounded setting to show an analogue of the P/poly inclusion. While demonstrating this discrepancy we show several other ways in which reductions to R_K and R_{K^t} are actually very different; in particular, we construct a counter-intuitive example of a polynomial-time non-adaptive reduction that distinguishes R_K from R_{K^t}, for any sufficiently large time-bound t.

To investigate the resource-bounded setting we define a class TTRT as a time-bounded analog of DTTR; informally, TTRT is the class of problems that are polynomial-time truth-table reducible to R_{K^t} for every sufficiently fast-growing time-bound t, and every "time-efficient" universal Turing machine used to define K^t. We prove that, for all monotone nondecreasing computable functions $\alpha(n) = \omega(1)$,

$$\text{BPP} \subseteq \mathsf{TTRT} \subseteq \text{PSPACE}/\alpha(n) \cap \text{P/poly}.$$

Here, PSPACE$/\alpha(n)$ is a "slightly non-uniform" version of PSPACE. We believe that this indicates that TTRT is "closer" to BPP than it is to PSPACE.

It would be more appealing to avoid the advice function, and we are able to do so, although this depends on a fine point in the definition of time-efficient prefix-free Kolmogorov complexity. This point involves a subtle technical distinction, and will be left for the appropriate section. To summarize:

- In Section 3 we prove that TTRT $\subseteq$ P/poly, by using the same basic idea of [5,1]. We further show, however, that this approach will not work to prove DTTR $\subseteq$ P/poly, and by reversing the logic connection of [5,1], this will give us an independence result in certain extensions of Peano arithmetic.
- Then in section 4 we prove that TTRT $\subseteq$ PSPACE$/\alpha(n)$, which is a non-trivial adaptation of the techniques from [6]. In section 5 we show how to get the result without the super-constant advice term.

In the final section we discuss prospects for future work.

2 Preliminaries

We assume the reader is familiar with basic complexity theory [7] and Kolmogorov complexity [12]. We use $\leq_T^p$ and P^A when referring to polynomial-time Turing reductions, and $\leq_{tt}^{p}$ and P_{tt}^{A} for polynomial-time truth-table (or *non-adaptive*) reductions. For example, $M : A \leq_{T}^{p} B$ means that M is a Turing reduction from A to B. For a set A of strings, $A^{\leq n}$ denotes the set of all strings of length at most n in A.

We let K_U denote Kolmogorov complexity with respect to prefix machine U, i.e., $K_U(x) = \min\{|p| : U(p) = x\}$. We use R_{K_U} to denote the set of K_U-*random* strings $\{x | K_U(x) \geq |x|\}$. In this paper, a function $t : \mathbb{N} \to \mathbb{N}$ is called a "time-bound" if it is non-decreasing and time-constructible. We use the following time-bounded version of Kolmogorov complexity: for a prefix machine U and a time-bound t, $K_U^t(x)$ is the length of the smallest string p such that $U(p)$ outputs x and halts in fewer than $t(|x|)$ time steps. Then $R_{K_U^t}$ is the set of K_U^t-*random* strings $\{x | K_U^t(x) \geq |x|\}$. Let us define what it means for a machine to be "universal" in the time-bounded setting:

Definition 2.1. *A prefix machine U is a time-efficient universal prefix machine if there exist constants c and c_M for each prefix machine M, such that*

1. *$\forall x$, $K_U(x) \leq K_M(x) + c_M$*
2. *$\forall x$, $K_U^t(x) \leq K_M^{t'}(x) + c_M$ for all $t > t'^c$*

We will sometimes omit U in the notation K_U, R_{K_U}, K_U^t, $R_{K_U^t}$, in which case we mean $U = U_0$, for some arbitrary choice of a time-efficient universal prefix machine U_0. Now we can formally define the time-bounded analogue of DTTR:

Definition 2.2. TTRT *is the class of languages L such that there exists a time bound t_0 (depending on L) such that for all time-efficient universal prefix machines U and $t \geq t_0$, $L \leq_{tt}^{p} R_{K_U^t}$.*

Corollary 12 from [8] says that, if $t \geq t_0 = 2^{2^{2n}}$, then BPP $\leq_{tt}^{p} R_{K_U^t}$, for any time-efficient universal U. This implies:

Theorem 2.3 ([8]). BPP $\subseteq$ TTRT.

Proposition 2.4. *For any machine M and $t'(|x|) > 2^{|x|}t(|x|)$, the query $x \in R_{K_M^t}$? can be computed in time t'.*

Due to page limits, we refer the reader to [2] for this proof and several other proofs that are omitted here.

Proposition 2.5. *Let $L \leq_{tt}^{p} R_{K_U^t}$ for some time-bound t. Then there exists a constant k such that the language L can be computed in $t_L(n) = 2^{n^k} t(n^k)$ time.*

It is the ability to compute R_{K^t} for short strings that makes the time-bounded case different from the ordinary case. This will be seen in proofs throughout the paper.

3 How and Why to Distinguish R_K from R_{K^t}

At first glance, it seems reasonable to guess that a polynomial-time reduction would have difficulty telling the difference between an oracle for R_K and an oracle for R_{K^t}, for large enough t. Indeed $R_K \subseteq R_{K^t}$ and in the limit for $t \to \infty$ they coincide.

One might even suspect that a polynomial-time reduction must behave the same way with R_{K^t} and R_K as oracle, already for modest time bounds t. However, this intuition is wrong. Here is an example for adaptive polynomial-time reductions.

Observation 3.1. *There is a polynomial-time algorithm which, given oracle access to R_K and input 1^n, outputs a K-random string of length n. However, for any time-bound t such that $t(n+1) \gg 2^n t(n)$, there is no polynomial-time algorithm which, given oracle access to R_{K^t} and input 1^n, outputs a K^t-random string of length n.*

For the algorithm, see [9]; roughly, we start with a small random string and then use [9, Theorem 15] (described later) to get a successively larger random string. But in the time-bounded case in [10] it is shown that on input 1^n, no polynomial-time machine M can query (or output) any K^t-random string of length n: in fact, $M(1^n)$ is the same for both oracles R_{K^t} and $R' = R_{K^t}^{\leq n-1}$. This is proven as follows: since R' can be computed in time $t(n)$ (by Proposition 2.4), then any query of length $\geq n$ made by $M^{R'}(1^n)$ is described by a pointer of length $O(\log n)$ in time $t(n)$, and hence is not in R_{K^t}.

3.1 Small Circuits for Sets Reducible to R_{K^t}

We now prove that TTRT is a subset of P/poly. Actually, we will prove that this holds even for Turing reductions to $R_{K_U^t}$ for a single universal Turing machine U (for large enough t):

Theorem 3.2. *Suppose $A \in \mathsf{DTIME}(t_1)$ and $M : A \leq_T^p R_{K^t}$, for some time-bounds t, t_1 with $t(n+1) \geq 2^n t(n) + 2^{2^n} t_1(2^n)$.*[1] *Then $A \in$ P/poly; in fact, if M runs in time n^c, and $R' = R_{K^t}^{\leq \lceil (c+1)\log n \rceil}$, then $\forall x \in \{0,1\}^n \; M^{R'}(x) = A(x)$.*

Proof. Let $\ell(n) = \lceil (c+1)\log n \rceil$, $R'(n) = R_{K^t}^{\leq \ell(n)}$, and suppose that $M^{R'(n)}(x) \neq A(x)$ for some x of length n. Then we may find the first such x in time $2^{\ell(n)} t(\ell(n)) + 2^{n+1}(t_1(n) + O(n^c))$ (cf. Proposition 2.4), and each query made by $M^{R'(n)}(x)$ can be output by a program of length $c \log n + O(1)$, running in the same time bound. But since $A(x) \neq M^{R'(n)}(x)$, it must be that, with $R'(n)$ as oracle, M makes some query q of size $m \geq \ell(n) + 1$ which is random for t-bounded Kolmogorov complexity (because both small and non-random queries are answered correctly when using R' instead of R_{K^t}). Hence we have both that q is supposed to be random, and that q can be output by a program of length $< \ell(n)$ in time $2^{\ell(n)} t(\ell(n)) + 2^{n+1}(t_1(n) + O(n^c)) \ll 2^{\ell(n)} t(\ell(n)) + 2^{2^{\ell(n)}} t_1(2^{\ell(n)}) \leq t(\ell(n)+1) \leq t(m)$, which is a contradiction. □

Corollary 3.3. $\mathsf{TTRT} \subseteq$ P/poly

PSPACE $\leq_T^p R_K$ [4], but Theorem 3.2 implies that PSPACE $\not\leq_T^p R_{K^t}$ for sufficiently-large t, unless PSPACE $\subseteq$ P/poly. This highlights the difference between the time-bounded and ordinary Kolmogorov complexity, and how this comes to the surface when working with reductions to the corresponding sets of random strings.

[1] For example, if $A \in$ EXP, then t can be doubly-exponential. If A is elementary-time computable, then t can be an exponential tower.

3.2 A Reduction Distinguishing R_K from R_{K^t}, and an Incorrect Conjecture

Theorem 3.2 shows that a polynomial-time truth-table reduction to R_{K^t} for sufficiently-large t will work just as well if only the logarithmically-short queries are answered correctly, and all of the other queries are simply answered "no".

The authors of [5] conjectured that a similar situation would hold if the oracle were R_K instead of R_{K^t}. More precisely, they proposed a proof-theoretic approach towards proving that DTTR is in P/poly: Let PA$_0$ denote Peano Arithmetic, and for $k > 0$ let PA$_k$ denote PA$_{k-1}$ augmented with the axiom "PA$_{k-1}$ is consistent". In [5] it is shown that, for any polynomial-time truth-table reduction M reducing a decidable set A to R_K, one can construct a true statement of the form $\forall n \forall j \forall k \Psi(n, j, k)$ (which is provable in a theory such as Zermelo-Frankel), with the property that if, for each fixed (**n**,**j**,**k**) there is some k' such that PA$_{k'}$ proves ψ(**n**,**j**, **k**), then DTTR $\subseteq$ P/poly. Furthermore, if these statements were provable in the given extensions of PA, it would follow that, for each input length n, there is a finite subset $R' \subseteq R_K$ consisting of strings having length at most $O(\log n)$, such that $M^{R'}(x) = A(x)$ for all strings x of length n.

Thus the authors of [5] implicitly conjectured that, for any polynomial-time truth-table reduction of a decidable set to R_K, and for any n, there would be some setting of the short queries so that the reduction would still work on inputs of length n, when all of the long queries are answered "no". While we have just seen that this is precisely the case for the time-bounded situation, the next theorem shows that this does not hold for R_K, even if "short" is interpreted as meaning "of length $< n$". (It follows that infinitely many of the statements ψ(**n**,**j**, **k**) of [5] are independent of every PA$_{k'}$.)

Theorem 3.4. *There is a truth-table reduction $M : \{0,1\}^* \leq_{tt}^p R_K$, such that, for all large enough n:*

$$\forall R' \subseteq \{0,1\}^{\leq n-1} \exists x \in \{0,1\}^n \; M^{R'}(x) \neq 1.$$

Proof. Theorem 15 of [9] presents a polynomial-time procedure which, given a string z of even length $n-2$, will output a list of constantly-many strings $z_1, \ldots, z_c$ of length n, such that at least one of them will be K-random if z is. We use this to define our reduction M as follows: on input $x = 00\ldots0z$ of length n having even $|z|$, we query each of z, $z_1, \ldots, z_c$, and every string of length at most $\log n$. If there are no strings of length at most $\log n$ in the oracle, we reject. Else, if z is in the oracle but none of the z_i are, we reject. On all other cases we accept.

By [9, Theorem 15], and since R_K has strings at every length, it is clear that M accepts every string with oracle R_K, and rejects every string if $R' = \varnothing$. However, for any non-empty set $R' \subseteq \{0,1\}^{\leq n-1}$, let $\ell \leq n-1$ be the highest even length for which $R'^{=\ell} \neq \varnothing$, and pick $z \in R'^{=\ell}$. Then we will have $z \in R'^{=\ell}$ but every $z_i \notin R^{=\ell+2}$, hence $M^{R'}(00\ldots0z)$ rejects. □

In fact, if we let $R' = R_{K^t}^{\leq n-1}$, for even n, then for the first $x = 00z$ such that $M^{R'}(x) = 0$, we will have $z \in R' \subseteq R_{K^t}$, but each z_i can be given by a small pointer in time $O(2^{n-1}t(n-1))$ (again we use Proposition 2.4), and hence $z_i \notin R_{K^t}$ for suitably fast-growing t. Thus $M^{R_{K^t}}(x) = 0 \neq M^{R_K}(x)$, and we conclude:

Observation 3.5. *If $t(n+1) \gg 2^n t(n)$, then the non-adaptive reduction M above behaves differently on the oracles R_K and R_{K^t}.*

4 Polynomial Space with Advice

Our single goal for this section is proving the following:

Theorem 4.1. *For any computable unbounded function $\alpha(n) = \omega(1)$,*

$$\mathsf{TTRT} \subseteq \text{PSPACE}/\alpha(n).$$

The proof of this theorem is patterned closely on related arguments in [6], although a number of complications arise in the time-bounded case. Because of space limitations, the presentation here will not be self-contained; readers will often be referred to [6].

Proposition 4.2 (Analogue to Coding Theorem). *Let f be a function such that*

1. $\sum_{x\in\{0,1\}^*} 2^{-f(x)} \leq 1$
2. *There is a machine M computing $f(x)$ in time $t(|x|)$*

Let $t'(|x|) > 2^{2|x|}t(|x|)$. Then for some M', $K^{t'}_{M'}(x) = f(x) + 2$.

Proposition 4.3 (Analogue to Proposition 6 from [6]). *Let U be a time-efficient universal prefix Turing machine and M be any prefix Turing machine. Suppose that t, t', and t'' are time bounds and f, g are two time-constructible increasing functions, such that f is upper bounded by a polynomial, and $t''(|x|) = f(t(|x|)) = g(t'(|x|))$.*

Then there is a time-efficient universal prefix machine U' such that

$$K^{t''}_{U'}(x) = \min(K^t_U(x), K^{t'}_M(x)) + 1$$

Proposition 4.4 (Analogue of Proposition 7 from [6]). *Given any time-efficient universal prefix machine U, time bound t, and constant $c \geq 0$, there is a time-efficient universal prefix machine U' such that $K^t_{U'}(x) = K^t_U(x) + c$.*

Proof (of Theorem 4.1). Fix α, and suppose for contradiction that $L \in \mathsf{TTRT} - \text{PSPACE}/\alpha(n)$. Let t_0 be the time bound given in the definition of TTRT, and let U_0 be some arbitrary time-efficient universal prefix machine. By the definition of TTRT, $L \leq^p_{tt} R_{K^{t_0}_{U_0}}$. Therefore, by Proposition 2.5, L is decidable in time $t_L(n) = 2^{n^k}t_0(n^k)$ for some constant k.

Let $t^*(n)$ be an extremely fast-growing function, so that for any constant d, we have $t^*(\log(\alpha(n))) > 2^{n^d}t_L(n)$ for all large n. To get our contradiction, we will show that there exists a time-efficient universal prefix machine U such that $L \not\leq^p_{tt} R_{K^{t^{*3}}_U}$. Note that because $t^* > t_0$, this is a contradiction to the fact that $L \in \mathsf{TTRT}$.

For any function $f : \{0,1\}^* \to \mathbb{N}$, define $R_f = \{x : f(x) \geq |x|\}$. We will construct a function $F : \{0,1\}^* \to \mathbb{N}$ and use it to form a function $H : \{0,1\}^* \to \mathbb{N}$ such that:

1. F is a total function and $F(x)$ is computable in time ${t^*}^2(|x|)$ by a machine M.
2. $H(x) = \min(K^{t^*}_{U_0}(x) + 5, F(x) + 3)$.
3. $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq 1/8$
4. $L \not\leq^p_{tt} R_H$

Claim (Analogue of Claim 1 from [6]). Given the above properties $H = K_U^{t^{*3}}$ for some efficient universal prefix machine U.

By Property 4 this ensures that the theorem holds.

Proof. By Property 3 we have that $\sum_{x\in\{0,1\}^*} 2^{-(F(x)+3)} \leq 1/8$. Hence $\sum_{x\in\{0,1\}^*} 2^{F(x)} \leq 1$. Using this along with Property 1, we then have by Proposition 4.2 that $K_{M'}^{t^{*3}} = F + 2$ for some prefix machine M'. By Proposition 4.4 we have that $K_{U'}^{t^*} = K_{U_0}^{t^*} + 4$ for some efficient universal prefix machine U'. Therefore, by Proposition 4.3, with $f(n) = n^3, g(n) = n$, we find that $H(x) = \min(K_{U_0}^{t^*}(x) + 5, F(x) + 3) = \min(K_{U'}^{t^*}(x), K_{M'}^{t^{*3}}) + 1$ is $K_U^{t^{*3}}$ for some efficient universal prefix machine U. □

We now need to show that, for our given language L, we can always construct functions H and F with the desired properties. As part of this construction we will set up and play a number of games. Our moves in the game will define the function F. Potentially during one of these games, we will play a move forcing a string z to be in the complement of R_H. To do this we will set $F(z) = |z| - 4$. Therefore, a machine M can compute $F(z)$ by running our construction, looking for the first time during the construction that $F(z)$ is set to $|z| - 4$, and outputting $|z| - 4$. If a certain amount of time elapses during the construction without $F(z)$ ever being set to $|z| - 4$, then the machine M outputs the default value $2|z|$.

As in [6], to ensure that $L \not\leq_{tt}^{p} R_H$, we need to satisfy an infinite list of requirements of the form

R_e : γ_e is not a polynomial-time truth-table reduction of L to R_H.

In contrast to the situation in [6], we do not need to worry about playing different games simultaneously or dealing with requirements in an unpredictable order; we will first satisfy R_1, then R_2, etc. To satisfy R_e we will set up a game $\mathcal{G}_{e,x}$ for an appropriate string x of our choice, and then play out the game in its entirety. We will choose x so that we can win the game $\mathcal{G}_{e,x}$, which will ensure that R_e is satisfied. If the K player cheats on game $\mathcal{G}_{e,x}$, then we play $\mathcal{G}_{e,x'}$ for some x'. For the same reasons as in [6] the K player cannot cheat infinitely often on games for a particular e, so eventually R_e will be satisfied.

A game $\mathcal{G}_{e,x}$ will be played as follows:

First we calculate the circuit $\gamma_{e,x}$, which represents the reduction γ_e on input x. Let F^* be the function F as it is at this point of the construction when the game $\mathcal{G}_{e,x}$ is about to be played. For any query z_i that is an input of this circuit such that $|z_i| \leq \log(\alpha(|x|)) - 1$, we calculate $r_i = \min(K_{U_0}^{t^*}(z_i) + 5, F^*(z_i) + 3)$. If $r_i < |z_i|$ we substitute FALSE in for the query, and simplify the circuit accordingly, otherwise we substitute TRUE in for the query, and simplify the circuit accordingly. (We will refer to this as the "pregame preprocessing phase".)

The remaining queries z_i are then ordered by increasing length. There are two players, the F player (whose moves will be played by us during the construction), and the K player (whose moves will be determined by K_{U_0}). As in [6], in each game the F player will either be playing on the YES side (trying to make the final value of the circuit equal TRUE), or the NO side (trying to make the final value of the circuit equal FALSE).

Let S_1 be the set of queries from $\gamma_{e,x}$ of smallest length, let S_2 be the set of queries that have the second smallest length, etc. So we can think of the queries being grouped into an ordered set $\mathcal{S} = (S_1, S_2, \ldots, S_r)$ for some r.

The scoring for the game is similar to that in [6]; originally each player has a score of 0 and a player loses if his score exceeds some threshold ϵ. When playing a game $\mathcal{G}_{e,x}$, we set $\epsilon = 2^{-e-6}$.

In round one of the game, the K player makes some (potentially empty) subset Z_1 of the queries from S_1 nonrandom. For any $Z_1 \subseteq S_1$ that he chooses to make nonrandom, $\sum_{z \in Z_1} 2^{-(|z|-6)} - 2^{-2|z|}$ is added to his score. As in [6], a player can only legally make a move if doing so will not cause his score to exceed ϵ.

Let us provide some explanation of how to interpret this score. Originally the function H is set so that for all z, $H(z) = 2|z|$. Because $H = \min(K^{t^*}_{U_0} + 5, F + 3)$, if $K^{t^*}_{U_0}(z) \leq |z| - 6$ then this ensures that z will be non-random according to H. It would be sub-optimal for the K player to set $K^{t^*}_{U_0}(z)$ to a value lower than $|z| - 6$, because this would add more to his score without any additional benefit. Therefore we assume without loss of generality that when the K player makes a move he does so in exactly this way. Thus the amount that is added to the score of player K corresponds to the amount by which K is changing the probability assigned to each string z (viewing K as a probability function). As in [6], for the case of analyzing the games and determining who has a winning strategy, we assume that the K player is an adversary playing optimally, even though in reality his moves will be based on an enumeration that knows nothing of these games.

After the K player makes his move in round 1, the F player responds, by making some subset Y_1 of the queries from $S_1 - Z_1$ nonrandom. After the F player moves, $\sum_{z \in Y_1} 2^{-(|z|-4)} - 2^{-2|z|}$ is added to his score.

This is the end of round one. Then we continue on to round two, played in the same way. The K player goes first and makes some subset of the queries from S_2 nonrandom (which makes his score go up accordingly), and then the F player responds by making some subset of the remaining queries from S_2 nonrandom. Note that if a query from S_i is not made nonrandom by either the K player or the F player in round i, it cannot be made nonrandom by either player for the remainder of the game.

After r rounds are finished the game is done and we see who wins, by evaluating the circuit $\gamma_{e,x}$ using the answers to the queries that have been established by the play of the game. If the circuit evaluates to TRUE (FALSE) and the F player is playing as the YES (NO) player, then the F player wins, otherwise the K player wins.

Note that the game is asymmetric between the F player and the K player; the F player has an advantage due to the fact that he plays second in each round and can make an identical move for fewer points than the K player. Because the game is asymmetric, it is possible that F can have a winning strategy playing on *both* the YES and NO sides. Thus we define a set $val(\mathcal{G}_{e,x'}) \subseteq \{0, 1\}$ as follows: $0 \in val(\mathcal{G}_{e,x'})$ if the F player has a winning strategy playing on the NO side in $\mathcal{G}_{e,x'}$, and $1 \in val(\mathcal{G}_{e,x'})$ if the F player has a winning strategy playing on the YES side in $\mathcal{G}_{e,x'}$.

Now we describe the construction. Suppose s time steps have elapsed during the construction up to this point, and we are getting ready to construct a new game in order to satisfy requirement R_e. (Either because we just finished satisfying requirement R_{e-1}, or because K cheated on some game $\mathcal{G}_{e,x}$, so we have to start a new game $\mathcal{G}_{e,x'}$).

Starting with the string $0^{{t^*}^4(s)}$ (i.e. the string of ${t^*}^4(s)$ zeros), we search strings in lexicographical order (as we do in [6]) until we find an x' such that $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. (Here, L denotes the characteristic function of the set L.)

Once we find such a string x' (which we will prove we always can), then we play out the game $\mathcal{G}_{e,x'}$ with the F player (us) playing on the YES side if $L(x') = 0$ and the NO side if $L(x') = 1$. To determine the K player's move in the ith round, we let $Z_i \subseteq S_i$ be those queries $z \in S_i$ for which $K^{t^*}_{U_0}(z) + 5 < |z|$. Our moves are determined by our winning strategy, and are played as in [6]. (These determine the function F; as in [6] initially $F(x) = 2|x|$ for all x). If the game is completed without the K player cheating, then we will have won the game, and R_e will be satisfied and will stay satisfied for the rest of the construction.

Note that when a game $\mathcal{G}_{e,x}$ is played, x is always chosen large enough so that any query that is not fixed during the pregame preprocessing has not appeared in any game that was played previously, so the games will never conflict with each other.

The analysis for why Properties 3 and 4 hold is basically identical to [6].

To wrap up the proof of the theorem, we need to prove a couple of claims.

Claim (Analogue of Claim 4 from [6]). During the construction, for any requirement R_e, we can always find a witness x with the needed properties to construct $\mathcal{G}_{e,x}$.

Proof. Suppose for some requirement R_e, our lexicographical search goes on forever without finding an x such that $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. Then $L \in$ PSPACE$/\alpha(n)$, which is a contradiction.

Here is the PSPACE algorithm to decide L. Hardcode all the answers for the initial sequence of strings up to the point where we got stuck in the construction. Let F^* be the function F up to that point in the construction. On a general input x, construct $\gamma_{e,x}$. The advice function $\alpha(n)$ will give the truth-table of $\min(K^{t^*}_{U_0}(z) + 5, F^*(z) + 3)$ for all queries z such that $|z| \leq \log(\alpha(|x|)) - 1$. For any query z of $\gamma_{e,x}$ such that $|z| \leq \log(\alpha(|x|)) - 1$, fix the answer to the query according to the advice.

If the F player had a winning strategy for both the YES and NO player on game $\mathcal{G}_{e,x}$, then we wouldn't have gotten stuck on R_e. Also the F player must have a winning strategy for either the YES or the NO player, since he always has an advantage over the K player when playing the game. Therefore, because we got stuck, it must be that the F player has a winning strategy for the YES player if and only if $L(x) = 1$. Once the small queries have been fixed, finding which side (YES or NO) the F player has a winning strategy for on $\mathcal{G}_{e,x}$, and hence whether $L(x) = 1$ or $L(x) = 0$, can be done in PSPACE.[2] □

Claim. $F(z)$ is computable in time ${t^*}^2(|z|)$

Proof. The function F is determined by the moves we play in games during the construction. In order to prove the claim, we must show that if during the construction we

[2] This follows from [6], as these games are a restricted case of the games from that paper. The point is that we can write the predicate "The F player has a winning strategy as the YES player on $\mathcal{G}_{e,x}$" as a simple quantified boolean formula.

as the F player make a move that involves setting a string z to be non-random, then fewer than ${t^*}^2(|z|)$ time steps have elapsed during the construction up to that point. The machine M that computes F will on input z run the construction for ${t^*}^2(|z|)$ steps. If at some point before this during the construction we as the F player make z non-random, then M outputs $|z| - 4$. Otherwise M outputs $2|z|$.

Suppose during the construction that we as the F player make a move that sets a query z to be non-random during a game $\mathcal{G}_{e,x}$. Note that $|z| \geq \log(\alpha(|x|))$, otherwise z would have been fixed during the preprocessing stage of the game.

There are at most $2^{|x|+1}$ strings x' that we could have considered during our lexicographic search to find a game for which we had a winning strategy before finally finding x. Let s be the number of time steps that have elapsed during the construction before this search began.

Let us first bound the amount of time it takes to reject each of these strings x'. To compute the circuit $\gamma_{e,x'}$ takes at most $|x'|^k$ time for some constant k. For each query y such that $|y| \leq \log(\alpha(|x'|)) - 1$ we compute $\min(K^{t^*}_{U_0}(y) + 5, F^*(y) + 3)$. To calculate $F^*(y)$ it suffices to rerun the construction up to this point and check whether a move had been previously made on the string y. To do this takes s time steps, and by construction we have that $t^*(|z|) \geq t^*(\log\alpha(|x|)) > |x'| \geq {t^*}^4(s)$, so $s < |z|$. By Proposition 2.4, to compute $K^{t^*}_{U_0}(y)$ takes at most $2^{|y|}t^*(|y|) \leq 2^{|z|}t^*(|z|)$ times steps. Therefore, since there can be at most $|x'|^k$ such queries, altogether computing $\min(K^{t^*}_{U_0}(y) + 5, F^*(y) + 3)$ for all these y will take fewer than $|x'|^k 2^{|z|}t^*(|z|)$ time steps.

Then we must compute $L(x')$, and check whether $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. Computing $L(x')$ takes $t_L(|x'|)$ time. By Claim 4, once the small queries have been fixed appropriately, computing $val(\mathcal{G}_{e,x'})$ can be done in PSPACE, so it takes at most $2^{|x'|^d}$ time for some constant d.

Compiling all this information, and using the fact that for each of these x' we have that $|x'| \leq |x|$, we get that the total number of timesteps needed to reject all of these x' is less than $2^{|x|^{d'}} 2^{|z|} t_L(|x|)t^*(|z|)$ for some constant d'.

During the actual game $\mathcal{G}_{e,x}$, before z is made non-random the construction might have to compute $K^{t^*}_{U_0}(y) + 5$ for all queries of $\gamma_{e,x}$ for which $|y| \leq |z|$. By Proposition 2.4 this takes at most $|x|^k 2^{|z|}t^*(|z|)$ time.

Therefore, overall, for some constant d'' the total amount of time steps elapsed before z is made non random in the construction is at most

$$T = 2^{|x|^{d''}} 2^{|z|} t_L(|x|)t^*(|z|) + s < {t^*}^2(|z|).$$

Here the inequality follows from the fact that $t^*(\log(\alpha(|x|))) > 2^{|x|^d} t_L(|x|)$ for any constant d, and that $|z| \geq \log(\alpha(|x|))$. □

5 Removing the Advice

With the plain Kolmogorov complexity function C, it is fairly clear what is meant by a "time-efficient" universal Turing machine. Namely, U is a time-efficient universal Turing machine if, for every Turing machine M, there is a constant c so that, for every

x, if there is a description d for which $M(d) = x$ in t steps, then there is a description d' of length $\leq |d| + c$ for which $U(d') = x$ in at most $ct \log t$ steps. However, with prefix-free Kolmogorov complexity, the situation is more complicated. The easiest way to define universal Turing machines for the prefix-free Kolmogorov complexity function K is in terms of *self-delimiting Turing machines.* These are machines that have one-way access to their input tape; x is a valid input for such a machine if the machine halts while scanning the last symbol of x. For such machines, the notion of time-efficiency carries over essentially unchanged. However, there are several other ways of characterizing K (such as in terms of partial-recursive functions whose domains form a prefix code, or in terms of prefix-free entropy functions). The running times of the machines that give short descriptions of x using some of these other conventions can be substantially less than the running times of the corresponding self-delimiting Turing machines. This issue has been explored in detail by Juedes and Lutz [11], in connection with the P versus NP problem. Given that there is some uncertainty about how best to define the notion of time-efficient universal Turing machine for K^t-complexity, one possible response is simply to allow much more leeway in the time-efficiency requirement.

If we do this, we are able to get rid of the small amount of non-uniformity in our PSPACE upper bound.

Definition 5.1. *A prefix machine U is an f-efficient universal prefix machine if there exist constants c_M for each prefix machine M, such that*

1. $\forall x$, $K_U(x) \leq K_M(x) + c_M$
2. $\forall x$, $K_U^t(x) \leq K_M^{t'}(x) + c_M$ *for all* $t(n) > f(t'(n))$

In Definition 2.1 we defined a time-efficient universal prefix machine to be any poly(n)-efficient universal prefix machine.

Definition 5.2. *Define TTRT' to be the class of languages L such that for all computable f there exists t_0 such that for all f-efficient universal prefix machines U and $t \geq t_0$, $L \leq_{tt}^{p} R_{K_U^t}$.*

Theorem 5.3. BPP $\subseteq \mathsf{TTRT}' \subseteq$ PSPACE $\cap$ P/poly.

Note that $\mathsf{TTRT}' \subseteq \mathsf{TTRT}$, so from Theorem 3.2 we get $\mathsf{TTRT}' \subseteq$ P/poly. Also, the proofs in [8] can be adapted to show that BPP $\subseteq \mathsf{TTRT}'$. We refer the reader to [2] for the PSPACE inclusion.

6 Conclusion

We have made some progress towards settling our research question in the case of time-bounded Kolmogorov complexity, but we have also discovered that this situation is substantially different from the ordinary Kolmogorov complexity. Solving this latter case will likely prove to be much harder.

We would like to prove an exact characterization, such as BPP $= \mathsf{DTTR}$ (or the time-bounded analogue thereof), but there seems to be no naive way of doing this. It has been shown in [8] that the initial segment $R_K^{\leq \log n}$, a string of length n, requires circuits of size n/c, for some $c > 1$ and all large n; it is this fact that is used to simulate BPP. However, much stronger circuit lower bounds for the initial segment do not seem to hold (cf. Theorems 4–9 of [8]), suggesting that R_K has some structure. This structure can actually be detected — the reduction M of Theorem 3.4 can be adapted to distinguish R_K from a random oracle w.h.p. — but we still don't know of any way of using R_K non-adaptively, other than as a pseudo-random string. A new idea will be needed in order to either prove or disprove the BPP $=$ DTTR conjecture.

Acknowledgments. The first and third authors acknowledge NSF Grants CCF-0832787 and CCF-1064785.

References

1. Allender, E.: Curiouser and Curiouser: The Link between Incompressibility and Complexity. In: Cooper, S.B., Dawar, A., Löwe, B. (eds.) CiE 2012. LNCS, vol. 7318, pp. 11–16. Springer, Heidelberg (2012)
2. Allender, E., Buhrman, H., Friedman, L., Loff, B.: Reductions to the set of random strings:the resource-bounded case. Technical Report TR12-054, ECCC (2012)
3. Allender, E., Buhrman, H., Koucký, M.: What can be efficiently reduced to the Kolmogorov-random strings? Annals of Pure and Applied Logic 138, 2–19 (2006)
4. Allender, E., Buhrman, H., Koucký, M., van Melkebeek, D., Ronneburger, D.: Power from random strings. SIAM Journal on Computing 35, 1467–1493 (2006)
5. Allender, E., Davie, G., Friedman, L., Hopkins, S.B., Tzameret, I.: Kolmogorov complexity, circuits, and the strength of formal theories of arithmetic. Technical Report TR12-028, ECCC (2012) (submitted for publication)
6. Allender, E., Friedman, L., Gasarch, W.: Limits on the computational power of random strings. Information and Computation (to appear, 2012); special issue on ICALP 2011, See also ECCC TR10-139
7. Balcázar, J.L., Días, J., Gabarró, J.: Structural Complexity I. Springer (1988)
8. Buhrman, H., Fortnow, L., Koucký, M., Loff, B.: Derandomizing from random strings. In: 25th IEEE Conference on Computational Complexity (CCC), pp. 58–63. IEEE (2010)
9. Buhrman, H., Fortnow, L., Newman, I., Vereshchagin, N.K.: Increasing Kolmogorov Complexity. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 412–421. Springer, Heidelberg (2005)
10. Buhrman, H., Mayordomo, E.: An excursion to the Kolmogorov random strings. J. Comput. Syst. Sci. 54(3), 393–399 (1997)
11. Juedes, D.W., Lutz, J.H.: Modeling time-bounded prefix Kolmogorov complexity. Theory of Computing Systems 33(2), 111–123 (2000)
12. Li, M., Vitanyi, P.: Introduction to Kolmogorov Complexity and its Applications, 3rd edn. Springer (2008)

Approximate Graph Isomorphism*

Vikraman Arvind[1], Johannes Köbler[2],
Sebastian Kuhnert[2], and Yadu Vasudev[1]

[1] The Institute of Mathematical Sciences, Chennai, India
{arvind,yadu}@imsc.res.in
[2] Institut für Informatik, Humboldt-Universität zu Berlin, Germany
{koebler,kuhnert}@informatik.hu-berlin.de

Abstract. We study optimization versions of Graph Isomorphism. Given two graphs G_1, G_2, we are interested in finding a bijection π from $V(G_1)$ to $V(G_2)$ that maximizes the number of matches (edges mapped to edges or non-edges mapped to non-edges). We give an $n^{O(\log n)}$ time approximation scheme that for any constant factor $\alpha < 1$, computes an α-approximation. We prove this by combining the $n^{O(\log n)}$ time additive error approximation algorithm of Arora et al. [*Math. Program.,* 92, 2002] with a simple averaging algorithm. We also consider the corresponding minimization problem (of mismatches) and prove that it is NP-hard to α-approximate for any constant factor α. Further, we show that it is also NP-hard to approximate the maximum number of edges mapped to edges beyond a factor of 0.94.

We also explore these optimization problems for bounded color class graphs which is a well studied tractable special case of Graph Isomorphism. Surprisingly, the bounded color class case turns out to be harder than the uncolored case in the approximate setting.

1 Introduction

The graph isomorphism problem (GI for short) is a well-studied computational problem: Formally, given two graphs G_1 and G_2 on n vertices, decide if there exists a bijection $\pi\colon V(G_1) \to V(G_2)$ such that $(u, v) \in E_1$ iff $(\pi(u), \pi(v)) \in E_2$. It remains one of the few problems that are unlikely to be NP-complete and for which no polynomial time algorithm is known.

Though the fastest known graph isomorphism algorithm for general graphs has running time $2^{O(\sqrt{n \log n})}$ [6], polynomial-time algorithms are known for many interesting subclasses, e.g. bounded degree graphs [19], bounded genus graphs [21], and bounded eigenvalue multiplicity graphs [5].

Motivation and Related Work. In this paper we study a natural optimization problem corresponding to the graph isomorphism problem where the objective is to compute a bijection that *maximizes* the number of edges getting

* This work was supported by Alexander von Humboldt Foundation in its research group linkage program. The third author was supported by DFG grant KO 1053/7-1.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 100–111, 2012.

mapped to edges and non-edges getting mapped to non-edges. The main motivation for this study is to explore if approximate isomorphisms can be computed efficiently, given that the best known algorithm for computing exact isomorphisms has running time $2^{O(\sqrt{n \log n})}$. The starting point of our investigation is a well-known article of Arora, Frieze and Kaplan [2] in which they study approximation algorithms for a quadratic assignment problem based on randomized rounding. Among the various problems they study, they also observe that approximate graph isomorphisms between n vertex graphs can be computed up to *additive error* εn^2 in time $n^{O(\log n/\varepsilon^2)}$. We show that this algorithm can be modified to obtain a multiplicative error approximation scheme for the problem. However, when we consider other variants of approximate graph isomorphism, they turn out to be much harder algorithmically.

To the best of our knowledge, the only previous theoretical study of approximate graph isomorphism is this work of Arora, Frieze and Kaplan [2]. However, the problem of approximate isomorphism and more general notions of graph similarity and graph matching has been studied for several years by the pattern matching community; see e.g. the survey article [8]. That line of research is not really theoretical. It is based on heuristics that are experimentally studied without rigorous proofs of approximation guarantees. Similarly, the general problem of graph edit distance [10] also encompasses approximate graph isomorphism. Both graph matchings and graph edit distance give rise to a variety of natural computational problems that are well studied.

Optimization Versions of Graph Isomorphism. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two input graphs on the same number n of vertices. We consider the following optimization problems:

- $\mathsf{Max\text{-}EGI}$: Given G_1, G_2, find a bijection $\pi \colon V_1 \to V_2$ that maximizes the number of matched edges, i.e., $me(\pi) = \|\{(u,v) \in E_1 \mid (\pi(u), \pi(v)) \in E_2\}\|$.
- $\mathsf{Max\text{-}PGI}$: Given G_1, G_2, find a bijection $\pi \colon V_1 \to V_2$ that maximizes matched vertex pairs, i.e., $mp(\pi) = me(\pi) + \|\{(u,v) \notin E_1 \mid (\pi(u), \pi(v)) \notin E_2\}\|$.
- $\mathsf{Min\text{-}EGI}$: Given G_1, G_2, find a bijection $\pi \colon V_1 \to V_2$ that minimizes mismatched edges, i.e., $\overline{me}(\pi) = \|\{(u,v) \in E_1 \mid (\pi(u), \pi(v)) \notin E_2\}\|$.
- $\mathsf{Min\text{-}PGI}$: Given G_1, G_2, find a bijection $\pi \colon V_1 \to V_2$ that minimizes mismatched pairs, i.e., $\overline{mp}(\pi) = \overline{me}(\pi) + \|\{(u,v) \notin E_1 \mid (\pi(u), \pi(v)) \in E_2\}\|$.

As mentioned above, $\mathsf{Max\text{-}PGI}$ was studied before in [2]. $\mathsf{Max\text{-}EGI}$ can also be viewed as an optimization variant of subgraph isomorphism.

Clearly, $mp(\pi) + \overline{mp}(\pi) = \binom{n}{2}$ and $me(\pi) + \overline{me}(\pi) = \|E_1\|$. Thus solving one of the maximization problems with additive error is equivalent to solving the corresponding minimization problem with the same additive error. However, the minimization problems behave differently for multiplicative factor approximations, so we study them separately.

Bounded Color Class Graph Isomorphism. A natural restriction of GI is to vertex-colored graphs (G_1, G_2) where $V(G_1) = C_1 \uplus C_2 \uplus \ldots \uplus C_m$ and

$V(G_2) = C'_1 \uplus C'_2 \uplus \ldots \uplus C'_m$, and C_i, C'_i contain the vertices of G_1 and G_2, respectively, that are colored i. The problem is to compute a color-preserving isomorphism π between G_1 and G_2, i.e., an isomorphism π such that for any vertex u, u and $\pi(u)$ have the same color. The bounded color-class version GI_k of GI consists of instances such that $\|C_i\| = \|C'_i\| \leq k$ for all i. For GI_k, randomized [4] and deterministic [9] polynomial time algorithms are known.

It is, therefore, natural to study the optimization problems defined above in the setting of vertex-colored graphs where the objective function is optimized over all color-preserving bijections $\pi\colon V_1 \to V_2$. We denote these problems as $\mathsf{Max\text{-}PGI}_k$, $\mathsf{Max\text{-}EGI}_k$, $\mathsf{Min\text{-}PGI}_k$ and $\mathsf{Min\text{-}EGI}_k$, where k is a bound on the number of vertices having the same color.

Overview of the Results. We first recall the notion of an α-approximation algorithm for an optimization problem. We call an algorithm $\mathcal{A}$ for a maximization problem an α-approximation algorithm, where $\alpha < 1$, if given an instance $\mathcal{I}$ of the problem with an optimum $\mathsf{OPT}(\mathcal{I})$, $\mathcal{A}$ outputs a solution with value $\mathcal{A}(\mathcal{I})$ such that $\mathcal{A}(\mathcal{I}) \geq \alpha\mathsf{OPT}(\mathcal{I})$. Similarly, for a minimization problem, we say $\mathcal{B}$ is a β-approximation algorithm for $\beta > 1$, if for any instance $\mathcal{I}$ of the problem with an optimum $\mathsf{OPT}(\mathcal{I})$, $\mathcal{B}$ outputs a solution with value $\mathcal{B}(\mathcal{I})$ such that $\mathcal{B}(\mathcal{I}) \leq \beta\mathsf{OPT}(\mathcal{I})$.

Theorem 1. *For any constant $\alpha < 1$, there is an α-approximation algorithm for* $\mathsf{Max\text{-}PGI}$ *running in time $n^{O(\log n/(1-\alpha)^4)}$.*

We obtain the α-approximation algorithm for $\mathsf{Max\text{-}PGI}$ by combining the $n^{O(\log n)}$ time additive error algorithm of [2] with a simple averaging algorithm.

Next we consider the $\mathsf{Max\text{-}EGI}$ problem. Langberg et al. [17] proved that there is no polynomial-time $(1/2 + \varepsilon)$-approximation algorithm for the Maximum Graph Homomorphism problem for any constant $\varepsilon > 0$ assuming that a certain refutation problem has average-case hardness (for the definition and details of this assumption we refer the reader to [17]). We give a factor-preserving reduction from the Maximum Graph Homomorphism problem to $\mathsf{Max\text{-}EGI}$ thus obtaining the following result.

Theorem 2. *There is no $(\frac{1}{2} + \varepsilon)$-approximation algorithm for* $\mathsf{Max\text{-}EGI}$ *for any constant $\varepsilon > 0$ under the same average-case hardness assumption of [17].*

We observe that unlike in the case of GI_k, where polynomial time algorithms are known [4,9,20], in the optimization setting, these problems are computationally harder. We prove the following theorem by giving a factor-preserving reduction from $\mathsf{Max\text{-}2Lin\text{-}2}$ (e.g. see [16]) to $\mathsf{Max\text{-}PGI}_k$ and $\mathsf{Max\text{-}EGI}_k$.

Theorem 3. *For any $k \geq 2$,* $\mathsf{Max\text{-}PGI}_k$ *and* $\mathsf{Max\text{-}EGI}_k$ *are* NP*-hard to approximate beyond a factor of* 0.94*.*

Since, assuming the Unique Games Conjecture (UGC for short) of Khot [15], it is NP-hard to approximate $\mathsf{Max\text{-}2Lin\text{-}2}$ beyond a factor of 0.878 [16], the same bound holds under UGC for $\mathsf{Max\text{-}PGI}_k$ and $\mathsf{Max\text{-}EGI}_k$ by the same reduction. Since

Max-PGI$_k$ and Max-EGI$_k$ are easily seen to be instances of generalized 2CSP, they have constant factor approximation algorithms, for a constant factor depending on k. In fact, it turns out that Max-EGI$_2$ and Max-PGI$_2$ are tightly classified by Max-2Lin-2 with almost matching upper and lower bounds (details are given in Section 2). However, we do not know of similar gap-preserving reductions from general unique games (with alphabet size more than 2) to Max-PGI$_k$ or Max-EGI$_k$ for larger values of k.

The following results show that the complexity of Min-PGI and Min-EGI is significantly different from Max-PGI and Max-EGI.

Theorem 4. *There is no polynomial time approximation algorithm for* Min-PGI *with any multiplicative approximation guarantee unless* $\mathsf{GI} \in \mathsf{P}$.

Theorem 5. Min-PGI *does not have a PTAS unless* $\mathsf{P} = \mathsf{NP}$.

Theorem 6. *There is no polynomial time approximation algorithm for* Min-EGI *with any multiplicative approximation guarantee unless* $\mathsf{P} = \mathsf{NP}$.

Finally, we turn our attention to the minimization problems Min-PGI$_k$ and Min-EGI$_k$ on bounded color-class graphs. We prove that Min-PGI$_k$ is as hard as the minimization version of Max-2Lin-2, known in literature as the Min-Uncut problem, and that Min-EGI$_4$ is inapproximable for any constant factor unless $\mathsf{P} = \mathsf{NP}$ by reducing the Nearest Codeword Problem (NCP) to it.

Outline of the Paper. Our results on maximization problems are in Section 2, while Section 3 contains our results on the corresponding minimization problems. Section 4 concludes with some open problems.

2 Maximizing the Number of Matches

We first observe that computing optimal solutions to Max-PGI is NP-hard via a reduction from Clique.

Lemma 7. *Computing optimal solutions to* Max-PGI *instances is* NP*-hard.*

Proof. Let (G, k) be an instance of the Clique problem. Define the graphs $G_1 = G$ and $G_2 = K_k \cup \overline{K}_{n-k}$, i.e., a k-clique and $n - k$ isolated vertices. Let π_{opt} be a bijection that achieves the optimum value for this Max-PGI instance. Then G has a k-clique if and only if $mp(\pi_{opt}) = \binom{n}{2} - \|E_G\| + \binom{k}{2}$. □

Next we give a general method for combining an additive error approximation algorithm for Max-PGI with a simple averaging approximation algorithm to design an α-approximation algorithm for Max-PGI for any constant $\alpha < 1$.

Lemma 8. *Suppose $\mathcal{A}$ is an algorithm such that for any $\varepsilon > 0$, given a* Max-PGI *instance in form of two n-vertex graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, computes a bijection $\pi\colon V_1 \to V_2$ such that $mp(\pi) \geq \mathsf{OPT} - \varepsilon n^2$ in time $T(n, \varepsilon)$. Then there is an algorithm that computes for each $\alpha < 1$ an α-approximate solution for any* Max-PGI *instance (G_1, G_2) in time $O(T(n, (1-\alpha)^2/9) + n^3)$.*

Proof. Without loss of generality we can assume $V_1 = V_2 = [n]$. We denote the number of edges in G_i by t_i and the number of non-edges by $\overline{t}_i$. Notice that the optimum for $\mathsf{Max\text{-}PGI}$ satisfies $\mathsf{OPT} \leq t_1 + \overline{t}_2$. Let $\pi\colon [n] \to [n]$ be a permutation chosen uniformly at random. Then, an easy calculation shows that the expected number s of matched pairs is

$$s = \frac{t_1 t_2 + \overline{t}_1 \overline{t}_2}{\binom{n}{2}} = \frac{\binom{n}{2} - \overline{t}_2}{\binom{n}{2}} t_1 + \frac{\overline{t}_2}{\binom{n}{2}} \left(\binom{n}{2} - t_1 \right) = t_1 + \overline{t}_2 - \frac{2 t_1 \overline{t}_2}{\binom{n}{2}}.$$

It is not hard to see that one can deterministically compute a permutation σ such that $mp(\sigma) \geq s$; we defer this detail to the end of the proof. We now show how this algorithm can be combined with the additive error approximation algorithm $\mathcal{A}$ for $\mathsf{Max\text{-}PGI}$ to obtain an α-approximation algorithm for $\mathsf{Max\text{-}PGI}$. The combined algorithm distinguishes two cases based on the number of edges and non-edges in G_1 and G_2, respectively.

Case 1. ($\min\{t_1, \overline{t}_2\} \leq (1-\alpha)\binom{n}{2}/2$): In this case we compute a permutation σ with $mp(\sigma) \geq s$. Since

$$t_1 \overline{t}_2 = \max\{t_1, \overline{t}_2\} \min\{t_1, \overline{t}_2\} \leq \left(t_1 + \overline{t}_2\right) (1-\alpha) \binom{n}{2} / 2,$$

it follows that

$$s = t_1 + \overline{t}_2 - 2 t_1 \overline{t}_2 / \binom{n}{2} \geq \alpha \left(t_1 + \overline{t}_2\right) \geq \alpha \mathsf{OPT}.$$

Case 2. ($\min\{t_1, \overline{t}_2\} > (1-\alpha)\binom{n}{2}/2$): In this case we use algorithm $\mathcal{A}$ with $\varepsilon = (1-\alpha)^2/9$ to obtain a permutation π with $mp(\pi) \geq \mathsf{OPT} - \varepsilon n^2$. Since $t_1 + \overline{t}_2 + \overline{t}_1 + t_2 = 2\binom{n}{2}$, either $t_1 + \overline{t}_2 \leq \binom{n}{2}$ or $\overline{t}_1 + t_2 \leq \binom{n}{2}$. Without loss of generality assume $t_1 + \overline{t}_2 \leq \binom{n}{2}$ (otherwise we interchange G_1 and G_2), implying that either $t_1 \leq \binom{n}{2}/2$ or $\overline{t}_2 \leq \binom{n}{2}/2$. Further, since the expected value of $mp(\pi)$ when π is picked at random is $t_1 + \overline{t}_2 - 2 t_1 \overline{t}_2 / \binom{n}{2}$, it follows that for sufficiently large n,

$$\mathsf{OPT} \geq t_1 - t_1 \overline{t}_2 / \binom{n}{2} + \overline{t}_2 - t_1 \overline{t}_2 / \binom{n}{2} \geq \frac{\min\{t_1, \overline{t}_2\}}{2} > \frac{1-\alpha}{4} \binom{n}{2} \geq \frac{\varepsilon n^2}{1-\alpha}.$$

Hence, $mp(\pi) \geq \mathsf{OPT} - \varepsilon n^2 \geq \alpha \mathsf{OPT}$.

It remains to show how a permutation which achieves at least the expected number s of matched pairs can be computed deterministically. Suppose that $\sigma\colon [i] \to [n]$ is a *partial permutation.* Let $\pi\colon [n] \to [n]$ be a random permutation that extends σ, i.e., $\pi(j) = \sigma(j)$ for $j \in [i]$. Let $s(\sigma)$ denote the expected number of matched pairs over random permutations π that extend σ. It is easy to see that we can compute $s(\sigma)$ in polynomial time. We do this by counting the pairs in three parts: (a) pairs with both end points in $[i]$, (b) pairs with both end points in $[n] \setminus [i]$, and (c) pairs with one end point in $[i]$ and the other in $[n] \setminus [i]$. Matched pairs of type (a) depend only on σ and can be counted straightaway.

The expected number of matched pairs of type (b) is computed exactly as s above (since π restricted on $[n] \setminus [i]$ is random). The expected number of matched pairs of type (c) is given by $\sum_{j \in [i]} \frac{n_j n_{\sigma(j)} + (n-i-n_j)(n-i-n_{\sigma(j)})}{n-i}$, where n_j is the number of neighbors of j in the graph G_1 contained in $[n] \setminus [i]$ and $n_{\sigma(j)}$ is the number of neighbors of $\sigma(j)$ in the graph G_2 contained in $[n] \setminus \{\sigma(l) \mid l \in [i]\}$. The entire computation of $s(\sigma)$ takes $O(n^2)$ time.

Now, for $k \in [n] \setminus \{\sigma(l) \mid l \in [i]\}$, let $\sigma_k \colon [i+1] \to [n]$ denote the extension of σ by setting $\sigma(i+1) = k$. Since a random extension π of σ can map $i+1$ uniformly to any $k \in [n] \setminus \{\sigma(l) \mid l \in [i]\}$ it follows that

$$s(\sigma) = \frac{1}{n-i} \sum_k s(\sigma_k),$$

where the summation is over all $k \in [n] \setminus \{\sigma(l) \mid l \in [i]\}$.

Furthermore, each $s(\sigma_k)$ is efficiently computable, as explained above. Reusing partial computations, we can find k such that $s(\sigma_k) \geq s(\sigma)$ in time $O(n^2)$. Continuing thus, when we fix the permutation on all of $[n]$ we obtain a σ with $mp(\sigma) \geq s$ in $O(n^3)$ time. □

Note that any polynomial time additive ε-error algorithm for Max-PGI, i.e., an algorithm running in time $n^{\text{poly}(1/\varepsilon)}$ with an additive error $\leq \varepsilon n^2$, gives a polynomial time α-approximation algorithm for Max-PGI running in time $n^{\text{poly}(1/(1-\alpha))}$.

To complete the proof of Theorem 1, we formulate Max-PGI as an instance of a quadratic optimization problem called the Quadratic Assignment Problem (QAP for short) as was done in [2] and use an additive error approximation algorithm for the Quadratic Assignment Problem due to Arora, Frieze and Kaplan [2].

Given $\{c_{ijkl}\}_{1 \leq i,j,k,l \leq n}$, the Quadratic Assignment Problem is to find an $n \times n$ permutation matrix $x = (x_{ij})$ that maximizes $val(x) = \sum_{i,j,k,l} c_{ijkl} x_{ij} x_{kl}$. An instance of Max-PGI consisting of graphs $G_1 = ([n], E_1)$ and $G_2 = ([n], E_2)$ can be naturally expressed as a QAP instance by setting

$$c_{ijkl} = \begin{cases} 1 & \text{if } (i,k) \in E_1 \text{ and } (j,l) \in E_2 \text{ or } (i,k) \notin E_1 \text{ and } (j,l) \notin E_2 \\ 0 & \text{otherwise.} \end{cases}$$

This ensures that $val(x) = mp(\pi_x)$ for all permutation matrices x with corresponding permutation π_x; in particular, the optimum solutions of the Max-PGI and QAP instances achieve the same value.

There is no polynomial time α-approximation algorithm for QAP for any $\alpha < 1$ unless P = NP [2]. Arora, Frieze and Kaplan in [2] give a general quasi-polynomial time algorithm for QAP with an additive error. Formally, they prove the following theorem.

Theorem 9 ([2]). *There is an algorithm that, given an instance of* QAP *where each of the c_{ijkl} is bounded in absolute value by a constant c and given an ε, finds an assignment to x_{ij} such that $val(x) \geq val(x^*) - \varepsilon n^2$ where x^* is the assignment which attains the optimum. The algorithm runs in time $n^{O(c^2 \log n/\varepsilon^2)}$.*

Thus for the Max-PGI problem, using Theorem 9 we can find a permutation π such that $mp(\pi) \geq \mathsf{OPT} - \varepsilon n^2$ in time $n^{O(\log n/\varepsilon^2)}$. Combining this with Lemma 8, we get an α-approximation algorithm for Max-PGI running in time $n^{O(\log n/(1-\alpha)^4)}$ and this completes the proof of Theorem 1.

In contrast to the quasi-polynomial time approximation scheme for Max-PGI, we now show that Max-EGI is likely to be $(\frac{1}{2}+\varepsilon)$-hard to approximate. To this end, define the Maximum Graph Homomorphism problem (MGH) first studied in [17]. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, MGH asks for a mapping $\phi\colon V_1 \to V_2$ such that $\|\{(u,v) \in E_1 \mid (\phi(u), \phi(v)) \in E_2\}\|$ is maximized. Langberg et al. [17] proved that MGH is hard to approximate beyond a factor of $1/2+\varepsilon$ under a certain average case assumption. To prove Theorem 2, we give a factor-preserving reduction from MGH to Max-EGI.

Lemma 10. *There is a polynomial time algorithm that for a given* MGH *instance $\mathcal{I}$, constructs a* Max-EGI *instance $\mathcal{I}'$ with* $\mathsf{OPT}(\mathcal{I}) = \mathsf{OPT}(\mathcal{I}')$.

Proof. Given an MGH instance $\mathcal{I} = (G_1, G_2)$, we construct the Max-EGI instance $\mathcal{I}' = (G_1', G_2')$ as follows. The graphs G_1' and G_2' both have vertex set $V_1 \times V_2$. For each edge (u_1, v_1) in the graph G_1, we put a single edge between the vertices (u_1, w_2) and (v_1, w_2) in E_1', where w_2 is an arbitrary but fixed vertex in V_2, and for each edge (u_2, v_2) in the graph G_2, we put all $\|V_1\|^2$ edges between $V_1 \times \{u_2\}$ and $V_1 \times \{v_2\}$ in E_2'. It suffices to prove the following claim.

Claim. There is a mapping $\phi\colon V_1 \to V_2$ such that $\big\|\big\{(u,v) \in E_1 \mid (\phi(u), \phi(v)) \in E_2\big\}\big\| = k$ if and only if there is a permutation $\pi\colon V_1 \times V_2 \to V_1 \times V_2$ such that $\|\{(u,v) \in E_1' \mid (\pi(u), \pi(v)) \in E_2'\}\| = k$.

Given the mapping ϕ, we construct the permutation π as follows: For each $u_1 \in V_1$, π maps the vertex (u_1, w_2) of G_1' to the vertex $(u_1, \phi(u_1))$ in G_2'. The remaining $\|V_1\| \cdot \|V_2\| - \|V_1\|$ vertices of G_1' are mapped arbitrarily.

Then each edge $(u_1, v_1) \in E_1$ is satisfied by ϕ if and only if the corresponding edge between (u_1, w_2) and (v_1, w_2) in E_1' is satisfied by π. This follows from the fact that $(\phi(u_1), \phi(v_1)) \in E_2$ if and only there is an edge between $(u_1, \phi(u_1))$ and $(v_1, \phi(v_1))$ in E_2'.

Similarly, given a permutation π between G_1' and G_2', we can obtain a mapping $\phi\colon V_1 \to V_2$ achieving the same number of matched edges by letting $\phi(u_1) = v_2$, where v_2 is the second component of the vertex $\pi(u_1, w_2)$. □

Unlike in the case of Max-PGI, we observe that there cannot be constant factor approximation algorithms for Max-PGI$_k$ for all constants. This is in interesting contrast to the fact that GI for graphs with bounded color-class size is in P. We now prove the hardness of approximating Max-PGI$_k$ and Max-EGI$_k$ for any $k \geq 2$.

We prove the hardness by exhibiting a factor-preserving reduction from Max-2Lin-2, which is hard to approximate above a guarantee of 0.94 unless P = NP [13]. Given a set $E \subseteq \big\{x_i + x_j = b \mid i,j \in [n], b \in \{0,1\}\big\}$ of m equations over $\mathbb{F}_2$, the problem Max-2Lin-2 is to find an assignment to the variables $x_1, \ldots, x_n$ that maximizes the number of equations satisfied.

The following lemma proves the factor-preserving reduction from Max-2Lin-2 to Max-PGI$_k$. The proof for Max-EGI$_k$ is similar.

Lemma 11. *For any $k \geq 2$, there is a polynomial time algorithm that for a given* Max-2Lin-2 *instance $\mathcal{I}$ constructs a* Max-PGI$_{2k}$ *instance $\mathcal{I}'$ such that* $\mathsf{OPT}(\mathcal{I}') = (2k)^2\mathsf{OPT}(\mathcal{I})$.

Proof sketch. Let $E \subseteq \{x_i + x_j = b \mid i, j \in [n], b \in \{0,1\}\}$ be the equations of $\mathcal{I}$. As a first step, if there is a pair of equations $x_i + x_j = 1$ and $x_i + x_j = 0$ in E, remove both these equations and add a new equation $y_i + y_j = 1$ on two new variables y_i and y_j. Let E' be the new set of equations obtained. Notice that $\mathsf{OPT}(E) = \mathsf{OPT}(E')$. We now describe the construction of the instance $\mathcal{I}'$ of Max-PGI$_{2k}$. For each variable x_i, put two sets of vertices V_i^0 and V_i^1 with k vertices each of color i. Let $x_l + x_m = b$ be an equation in E'. In the graph G_1, add a complete bipartite graph between V_l^0 and V_m^0 and another complete bipartite graph between V_l^1 and V_m^1. Similarly, add the complete bipartite graph between V_l^0 and V_m^b and between V_l^1 and $V_m^{1\oplus b}$ in G_2. If there is no equation in E' connecting the variables x_l and x_m, add a complete bipartite graph between the color classes l and m in G_1 and the empty graph between l and m in G_2. Similarly, make all color classes cliques in G_1 and independent sets in G_2. The idea is that assigning $x_i \mapsto 0$ corresponds to mapping V_i^0 and V_i^1 to themselves, respectively, while assigning $x_i \mapsto 1$ corresponds to mapping V_i^0 to V_i^1 and vice versa. Because of space constraints, we omit the proof that this construction works; it can be found in [3]. □

This construction still works if we replace $mp(\pi)$ with $me(\pi)$, as for all equations $x_i + x_j = b$ in E, exactly half of the possible edges between color classes i and j are present. It follows that there is a factor-preserving reduction from Max-2Lin-2 to Max-EGI$_{2k}$.

Lemma 12. *For any $k \geq 2$, there is a polynomial time algorithm that for a given* Max-2Lin-2 *instance $\mathcal{I}$ constructs a* Max-EGI$_{2k}$ *instance $\mathcal{I}'$ such that* $\mathsf{OPT}(\mathcal{I}') = 2k^2\mathsf{OPT}(\mathcal{I})$.

Since there is no α-approximation algorithm for Max-2Lin-2 for $\alpha > 0.94$ unless $\mathsf{P} = \mathsf{NP}$ [13], Lemmas 11 and 12 complete the proof of Theorem 3 that there is no α-approximation algorithm for Max-PGI$_k$ and Max-EGI$_k$ for $\alpha > 0.94$ unless $\mathsf{P} = \mathsf{NP}$.

It is easy to see that for each constant $k > 0$, both Max-PGI$_k$ and Max-EGI$_k$ are subproblems of the generalized Max-2CSP(q), where q depends on k. Thus, both Max-PGI$_k$ and Max-EGI$_k$ have constant factor approximation algorithms by virtue of the semidefinite programming based approximation algorithm for Max-2CSP(q) [12]. The following lemma shows the reduction of Max-EGI$_2$ to Max-2CSP(2). The reduction from Max-PGI$_k$ and Max-EGI$_k$ to Max-2CSP(q) is similar.

Lemma 13. *There is a polynomial time algorithm that for two given vertex-colored graphs G_1 and G_2 where each color class has size at most 2, outputs a* Max-2CSP(2) *instance $\mathcal{F} = \{f_1, \ldots, f_m\}$ where $m = \|E(G_1)\|$ and $f_i \colon \{0,1\}^2 \to \{0,1\}$ such that there is a color-preserving bijection $\pi \colon V(G_1) \to V(G_2)$ with $me(\pi) = k$, if and only if there is an assignment which satisfies k constraints in $\mathcal{F}$.*

Proof. For each color class C_i, we assign a variable x_i. For an edge e from C_i to C_j in G_1, construct the function $f_e\colon \{0,1\}^2 \to \{0,1\}$ over the variables x_i and x_j as follows. Any Boolean assignment to the variables can be looked upon as a permutation: If $x_i \mapsto 0$, then we have the identity permutation on C_i, otherwise the permutation swaps the vertices of C_i. The value f_e on that particular assignment is 1 if the permutation that it corresponds to sends the edge e to an edge in G_2. Hence there is an assignment that satisfies k constraints if and only if there is a permutation π with $me(\pi) = k$. □

As the problem of Max-2CSP(2) has an approximation algorithm with a guarantee of 0.874 [18], this implies an approximation algorithm for Max-EGI$_2$ with the same guarantee and since Max-2Lin-2 is hard to approximate beyond 0.878 under UGC [16], we have almost matching upper and lower bounds for Max-EGI$_2$ under UGC.

3 Minimizing the Number of Mismatches

We first consider the problems Min-PGI and Min-EGI, where the objective is to minimize the number of mismatched pairs and edges, respectively.

Theorem 4. *There is no polynomial time approximation algorithm for* Min-PGI *with any multiplicative approximation guarantee unless* GI $\in$ P.

Proof. Assume that there is a polynomial time α-approximation algorithm $\mathcal{A}$ for Min-PGI. If the two input graphs G_1 and G_2 are isomorphic, then there is a bijection $\pi\colon V_1 \to V_2$ such that $\overline{mp}(\pi) = 0$, and if G_1 and G_2 are not isomorphic, then $\overline{mp}(\pi) > 0$ for all π. Thus, it immediately follows that G_1 and G_2 are isomorphic, if and only if $\mathcal{A}$ outputs a bijection $\sigma\colon V(G_1) \to V(G_2)$ with $\overline{mp}(\sigma) = 0$ (i.e., an isomorphism). □

In order to show that it is unlikely that Min-PGI has a polynomial time approximation scheme, we give a gap-preserving reduction from the Vertex-disjoint Triangle Packing problem (VTP) defined as follows: Given a graph G find the maximum number of vertex-disjoint triangles that can be packed into G. We look at the corresponding gap version of the VTP problem.

Gap-VTP$_{\alpha,\beta}$: Given a graph G and $\alpha > \beta$,

1. Answer YES, if at least $\alpha n/3$ triangles can be packed into G.
2. Answer NO, if at most $\beta n/3$ triangles can be packed into G.

It is known that VTP does not have an algorithm which when given a graph and parameter α as input, computes a vertex-disjoint triangle packing of size at least αOPT in time $O(n^{\mathrm{poly}(1/(1-\alpha))})$ unless P = NP [7]. It is also known that for a fixed value of $\beta < 1$, Gap-VTP$_{1,\beta}$ is NP-hard on graphs where each vertex is either degree 4 or 6 [11,22], and a small gadget shows that this also holds for 6-uniform graphs.

Lemma 14. *Given a* $\mathsf{Gap\text{-}VTP}_{\alpha,\beta}$ *instance* $\mathcal{I}$ *(a* 6*-uniform graph on* n *vertices), in polynomial time we can find a* Min-PGI *instance* $\mathcal{I}'$ *such that*

$$\mathsf{OPT}(\mathcal{I}) \geq \frac{\alpha n}{3} \Rightarrow \mathsf{OPT}(\mathcal{I}') \leq 2n(2-\alpha)$$
$$\mathsf{OPT}(\mathcal{I}) \leq \frac{\beta n}{3} \Rightarrow \mathsf{OPT}(\mathcal{I}') \geq \frac{2n}{3}(4-\beta)$$

The proof of this lemma is omitted here because of space constraints; it can be found in [3]. This reduction together with the hardness of VTP proves Theorem 5. Next we prove Theorem 6.

Theorem 6. *There is no polynomial time approximation algorithm for* Min-EGI *with any multiplicative approximation guarantee unless* P = NP*.*

Proof. The theorem follows from the following reduction from the Clique problem. Given an instance (G, k) of Clique, we construct the instance of Min-EGI as follows. G_1 consists of a k-clique and $n-k$ independent vertices, and $G_2 := G$. $(G, k) \in$ Clique if and only if there exists a π such that in the Min-EGI problem $\overline{me}(\pi) = 0$. Hence any polynomial time approximation algorithm with a multiplicative guarantee for Min-EGI gives a polynomial time algorithm for Clique. □

The input for the Min-Uncut problem is a set $E \subseteq \{x_i + x_j = 1 \mid i, j \in [n]\}$ of m equations. The objective is to minimize the number of equations that must be removed from the set E so that there is an assignment to the variables that satisfy all the equations. This problem is known to be MaxSNP-hard [14], and assuming the Unique Games Conjecture, hard to approximate within any constant factor [15]. The following lemma shows that $\mathsf{Min\text{-}PGI}_k$ is as hard as the Min-Uncut problem.

Lemma 15. *Let* $\mathcal{I}$ *be an instance of* Min-Uncut *and let* k *be a positive integer. There is a polynomial time algorithm that constructs an instance* $\mathcal{I}'$ *of* $\mathsf{Min\text{-}PGI}_{2k}$ *such that* $\mathsf{OPT}(\mathcal{I}') = (2k)^2\mathsf{OPT}(\mathcal{I})$*.*

The proof of this lemma is similar to the proof of Lemma 11. Given a set $E \subseteq \{x_i + x_j = 1 \mid i, j \in [n]\}$ of equations over $\mathbb{F}_2$, we construct an instance $\mathcal{I}'$ of $\mathsf{Min\text{-}PGI}_{2k}$ exactly as described in the proof of Lemma 11. If the minimum number of equations that have to be deleted from E to make the rest satisfiable is at most t, then there is an assignment such that at most t equations in E are not satisfied. This implies that there is a permutation π such that the only edges that are mapped to non-edges and vice-versa are from at most t pairs of color classes.

Finally we show that $\mathsf{Min\text{-}EGI}_4$ is hard to approximate.

Theorem 16. *For any constant* $\alpha > 1$*, there is no* α*-approximation algorithm for* $\mathsf{Min\text{-}EGI}_4$ *unless* P = NP*.*

An instance of NCP consists of a subspace $\mathcal{S}$ of $\mathbb{F}_2^n$ given as a set of basis vectors $\mathcal{B} = \{s_1, \ldots, s_k\}$ and a vector $v \in \mathbb{F}_2^n$. The objective is to find a vector $u \in \mathcal{S}$

which minimizes the hamming weight $\mathrm{wt}(u+v)$, i.e., the number of bits where u and v differ. It is NP-hard to approximate NCP within any constant factor [1]. The following lemma gives a reduction that transfers this hardness to $\mathsf{Min\text{-}EGI}_4$.

Lemma 17. *There is a polynomial time algorithm that for a given* NCP *instance* $\mathcal{I}$*, constructs a* $\mathsf{Min\text{-}EGI}_4$ *instance* $\mathcal{I}'$ *with* $\mathsf{OPT}(\mathcal{I}') = \mathsf{OPT}(\mathcal{I})$*.*

The idea of the proof is to construct two graphs G_1 and G_2 such that any vector from the given subspace $\mathcal{S}$ that is equal to v in all but k positions, can be converted into a color-preserving bijection from $V(G_1)$ to $V(G_2)$ that maps all but k edges to edges, and vice versa. A detailed proof is given in [3].

4 Conclusion

Although GI expressed as an optimization problem was mentioned in [2], as far as we know this is the first time that the complexity of the other three variants of this optimization problem has been studied. Considering the upper and lower complexity bounds that we have proved in this paper, the following questions seem particularly interesting.

In Theorem 1 we describe an α-approximation algorithm for $\mathsf{Max\text{-}PGI}$ that runs in quasi-polynomial time. Does $\mathsf{Max\text{-}PGI}$ also have a polynomial time approximation scheme? Theorem 2 shows that it is unlikely that $\mathsf{Max\text{-}EGI}$ has an $(\frac{1}{2}+\varepsilon)$-approximation algorithm. Does $\mathsf{Max\text{-}EGI}$ have a constant factor approximation algorithm? We can use the Quadratic Assignment Problem to get an additive error algorithm for it which runs in quasi-polynomial time but we do not know whether this algorithm can be used to get a constant factor approximation algorithm for $\mathsf{Max\text{-}EGI}$ (as was possible for $\mathsf{Max\text{-}PGI}$). In the case of vertex-colored graphs, even though we can rule out the existence of a PTAS for $\mathsf{Max\text{-}PGI}_k$ and $\mathsf{Max\text{-}EGI}_k$, it remains open whether these problems have efficient approximation algorithms providing a good constant factor approximation guarantee.

Acknowledgement. We thank the anonymous referees for their suggestions to improve the article.

References

1. Arora, S., Babai, L., Stern, J., Sweedyk, Z.: The hardness of approximate optima in lattices, codes, and systems of linear equations. J. Comput. Syst. Sci 54(2), 317–331 (1997)
2. Arora, S., Frieze, A.M., Kaplan, H.: A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. Math. Program. 92(1), 1–36 (2002)
3. Arvind, V., Köbler, J., Kuhnert, S., Vasudev, Y.: Approximate graph isomorphism. ECCC, TR12-078 (2012), http://eccc.hpi-web.de/report/2012/078/

4. Babai, L.: Monte-Carlo algorithms in graph isomorphism testing. Technical Report 79-10, Univ. de Montréal, Dép. de mathématiques et de statistique (1979)
5. Babai, L., Grigoryev, D.Y., Mount, D.M.: Isomorphism of graphs with bounded eigenvalue multiplicity. In: STOC, pp. 310–324 (1982)
6. Babai, L., Luks, E.M.: Canonical labeling of graphs. In: STOC, pp. 171–183 (1983)
7. Caprara, A., Rizzi, R.: Packing triangles in bounded degree graphs. Inf. Process. Lett. 84(4), 175–180 (2002)
8. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. IJPRAI 18(3), 265–298 (2004)
9. Furst, M.L., Hopcroft, J.E., Luks, E.M.: Polynomial-time algorithms for permutation groups. In: FOCS, pp. 36–41 (1980)
10. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Anal. Appl. 13(1), 113–129 (2010)
11. Guruswami, V., Indyk, P.: Embeddings and non-approximability of geometric problems. In: SODA, pp. 537–538 (2003)
12. Guruswami, V., Raghavendra, P.: Constraint Satisfaction over a Non-Boolean Domain: Approximation Algorithms and Unique-Games Hardness. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 77–90. Springer, Heidelberg (2008)
13. Håstad, J.: Some optimal inapproximability results. J. ACM 48(4), 798–859 (2001)
14. Khanna, S., Sudan, M., Trevisan, L., Williamson, D.P.: The approximability of constraint satisfaction problems. SIAM J. Comput. 30(6), 1863–1920 (2000)
15. Khot, S.: On the power of unique 2-prover 1-round games. In: STOC, pp. 767–775 (2002)
16. Khot, S., Kindler, G., Mossel, E., O'Donnell, R.: Optimal inapproximability results for max-cut and other 2-variable csps? In: FOCS, pp. 146–154 (2004)
17. Langberg, M., Rabani, Y., Swamy, C.: Approximation Algorithms for Graph Homomorphism Problems. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 176–187. Springer, Heidelberg (2006)
18. Lewin, M., Livnat, D., Zwick, U.: Improved Rounding Techniques for the MAX 2-SAT and MAX DI-CUT Problems. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 67–82. Springer, Heidelberg (2002)
19. Luks, E.M.: Isomorphism of graphs of bounded valence can be tested in polynomial time. J. Comput. Syst. Sci. 25(1), 42–65 (1982)
20. Luks, E.M.: Parallel algorithms for permutation groups and graph isomorphism. In: FOCS, pp. 292–302 (1986)
21. Miller, G.L.: Isomorphism of k-contractible graphs. a generalization of bounded valence and bounded genus. Information and Control 56(1/2), 1–20 (1983)
22. Petrank, E.: The hardness of approximation: Gap location. Computational Complexity 4, 133–157 (1994)

Near-Optimal Expanding Generator Sets for Solvable Permutation Groups

Vikraman Arvind[1], Partha Mukhopadhyay[2],
Prajakta Nimbhorkar[2], and Yadu Vasudev[1]

[1] The Institute of Mathematical Sciences, Chennai, India
{arvind,yadu}@imsc.res.in
[2] Chennai Mathematical Institute, Siruseri, India
{partham,prajakta}@cmi.ac.in

Abstract. Let $G = \langle S \rangle$ be a solvable subgroup of the symmetric group S_n given as input by the generator set S. We give a deterministic polynomial-time algorithm that computes an *expanding generator set* of size $\widetilde{O}(n^2)$ for G. As a byproduct of our proof, we obtain a new explicit construction of ε-bias spaces of size $\widetilde{O}(n\,\mathrm{poly}(\log d))(\frac{1}{\varepsilon})^{O(1)}$ for the groups $\mathbb{Z}_d^n$.

1 Introduction

Expander graphs are of great interest and importance in theoretical computer science, especially in the study of randomness in computation; the monograph by Hoory, Linial, and Wigderson [10] is an excellent reference. A central problem is the explicit construction of expander graph families [10,12]. By explicit it is meant that the family of graphs has efficient deterministic constructions, where the notion of efficiency depends upon the application, e.g. [14]. Explicit constructions with the best known, near optimal expansion and degree parameters (the so-called Ramanujan graphs) are Cayley expander families [12].

Alon and Roichman,in [4], show for any finite group G and $\lambda > 0$, that with high probability a multiset S of size $O(\frac{1}{\lambda^2}\log|G|)$ picked uniformly at random from G is a λ-spectral expander: I.e. the second largest eigenvalue in absolute value, of the normalized adjacency matrix of the Cayley graph is bounded by λ. If G is given by its multiplication table then there is a simple *Las Vegas* algorithm for computing S: pick a random multiset S of size $O(\frac{1}{\lambda^2}\log|G|)$ from G and check in deterministic time $|G|^{O(1)}$ that $\mathrm{Cay}(G,T)$ is a λ-spectral expander. Wigderson and Xiao give a deterministic polynomial-time algorithm for computing S by derandomizing this algorithm [17] (see [5] for a combinatorial proof).

This Paper

Suppose G, a subgroup of S_n, is given as input by a *generator set* S, and not a multiplication table. Can we compute an $O(\log|G|)$ size expanding generator set for G in deterministic time polynomial in n and $|S|$? Now, it is possible to

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 112–123, 2012.

sample nearly uniformly in polynomial time from G given by a generator set (e.g. see [8]). Therefore, by the Alon-Roichman theorem we have a randomized polynomial-time algorithm for the problem (although it is not Las Vegas since we do not know how to certify an expanding generator set in polynomial time).

This problem can be seen as a generalization of the construction of small bias spaces in $\mathbb{F}_2^n$ [3]. It is easily proved (see [10]) using properties of finite abelian groups, that ε-bias spaces are precisely the expanding generator sets for any finite abelian group. Interestingly, the best known explicit construction of ε-bias spaces is of size either $O(n^2/\varepsilon^2)$ [3] or $O(n/\varepsilon^3)$ [2], whereas the Alon-Roichman theorem guarantees the existence of ε-bias spaces of size $O(n/\varepsilon^2)$.

Subsequently, Azar, Motwani and Naor [7] gave a construction of ε-bias spaces for finite abelian groups of the form $\mathbb{Z}_d^n$ using Linnik's theorem and Weil's character sum bounds. The size of the ε-bias space they give is $O((d + n^2/\varepsilon^2)^C)$ where the current best known bound for C is 11/2.

Let G be a finite group, and let $S = \{g_1, g_2, \ldots, g_k\}$ be a *generating* set for G. The *undirected Cayley graph* $\mathrm{Cay}(G, S \cup S^{-1})$ is an undirected multigraph with vertex set G and edges of the form $\{x, xg_i\}$ for each $x \in G$ and $g_i \in S$. Since S is a generator set for G, $\mathrm{Cay}(G, S \cup S^{-1})$ is a connected regular multigraph.

In this paper we prove a more general result. Given any solvable subgroup G of S_n (where G is given by a generator set) and $\lambda > 0$, we construct an expanding generator set T for G of size $\widetilde{O}(n^2)(\frac{1}{\lambda})^{O(1)}$ such that $\mathrm{Cay}(G, T)$ is a λ-spectral expander. The exact constant factor in the upper bound is given in Section 4.

We also note that, for a *general* permutation group $G \leq S_n$ given by a generator set, we can compute (in deterministic polynomial time) an $O(n^c)(\frac{1}{\lambda})^{O(1)}$ size generator set T such that $\mathrm{Cay}(G, T)$ is a λ-spectral expander, where c is a large constant.

Now we explain the main ingredients of our expanding generator set construction for solvable groups.

1. Let G be a finite group and N be a normal subgroup of G. Given expanding generator sets S_1 and S_2 for N and G/N respectively such that the corresponding Cayley graphs are λ-spectral expanders, we give a simple polynomial-time algorithm to construct an expanding generator set S for G such that $\mathrm{Cay}(G, S)$ is also λ-spectral expander. Moreover, $|S|$ is bounded by a constant factor of $|S_1| + |S_2|$. The analysis in this section is similar to the work of [15,16].
2. We compute the derived series for the given solvable group $G \leq S_n$ in polynomial time using a standard algorithm [13]. This series is of $O(\log n)$ length due to Dixon's theorem. Let the derived series for G be $G = G_0 \rhd G_1 \rhd \cdots \rhd G_k = \{1\}$. Assuming that we already have an expanding generator set for each quotient group G_i/G_{i+1} (which is abelian) of size $\widetilde{O}(n^2)$, we apply the previous step repeatedly to obtain an expanding generator set for G of size $\widetilde{O}(n^2)$. We can do this because the derived series is a normal series.
3. Finally, we consider the abelian quotient groups G_i/G_{i+1} and give a polynomial time algorithm to construct expanding generator sets of size $\widetilde{O}(n^2)$ for them. This construction applies a series decomposition of abelian groups

as well as makes use of the Ajtai et al construction of expanding generator sets for $\mathbb{Z}_t$ [1]. This construction is motivated by work of Alon et al. [3].

We describe the steps 1, 2 and 3 outlined above in Sections 2, 3 and 4, respectively. As a simple application of our main result, we give a new explicit construction of ε-bias spaces for the groups $\mathbb{Z}_d^n$ which we explain in Section 5. The size of our ε-bias spaces are $O(n\,\mathrm{poly}(\log n, \log d))(\frac{1}{\varepsilon})^{O(1)}$. The known construction of ε-bias space for $\mathbb{Z}_d^n$ is of size $O((d + n/\varepsilon^2))^{11/2}$ [7]. The size bound of our construction improves on the Azar-Motwani-Naor construction in the parameters d and n.

It is interesting to ask, for a finite group G given by generator sets, whether we can obtain expanding generator sets of $O(\log |G|)$ size in deterministic polynomial time. By the Alon-Roichman theorem we know that such expanding generator sets for G exist.

In this connection, we note a negative result that Lubotzky and Weiss [11] have shown about solvable groups: Let $\{G_i\}$ be any infinite family of finite solvable groups $\{G_i\}$ where each G_i has derived series length bounded by some constant ℓ. Suppose Σ_i is any generator set for G_i of size $|\Sigma_i| \le k$ for each i and some constant k. Then the Cayley graphs $\mathrm{Cay}(G_i, \Sigma_i)$ do not form an expander family. In contrast, they also exhibit an infinite family of solvable groups in [11] that give rise to constant-degree Cayley expanders.

2 Combining Generator Sets for Normal Subgroup and Quotient Group

Let G be any finite group and N be a normal subgroup of G, denoted $G \rhd N$. I.e. $g^{-1}Ng = N$ for all elements $g \in G$. Let $A \subset N$ be an expanding generator set for N and $\mathrm{Cay}(N, A)$ be a λ-spectral expander. Similarly, let $B \subset G$ such that $\widehat{B} = \{Nx \mid x \in B\}$ is an expanding generator set for the quotient group G/N and $\mathrm{Cay}(G/N, \widehat{B})$ is λ-spectral. We first show that $\mathrm{Cay}(G, A \cup B)$ is a $\frac{1+\lambda}{2}$-spectral expander.

In order to analyze the spectral expansion of the Cayley graph $\mathrm{Cay}(G, A \cup B)$ it is useful to view vectors in $\mathbb{C}^{|G|}$ as elements of the group algebra $\mathbb{C}[G]$. The group algebra $\mathbb{C}[G]$ consists of linear combinations $\sum_{g\in G} \alpha_g g$ for $\alpha_g \in \mathbb{C}$. Addition in $\mathbb{C}[G]$ is component-wise, and clearly $\mathbb{C}[G]$ is a $|G|$-dimensional vector space over $\mathbb{C}$. The product of $\sum_{g\in G} \alpha_g g$ and $\sum_{h\in G} \beta_h h$ is defined naturally as: $\sum_{g,h\in G} \alpha_g \beta_h gh$.

Let $S \subset G$ be any symmetric subset (i.e. S is closed under inverse) and let M_S denote the normalized adjacency matrix of the undirected Cayley graph $\mathrm{Cay}(G, S)$. Now, each element $a \in G$ defines the linear map $M_a : \mathbb{C}[G] \to \mathbb{C}[G]$ by $M_a(\sum_g \alpha_g g) = \sum_g \alpha_g ga$. Clearly, $M_S = \frac{1}{|S|} \sum_{a\in S} M_a$ and $M_S(\sum_g \alpha_g g) = \frac{1}{|S|} \sum_{a\in S} \sum_g \alpha_g ga$.

Let $X = \{x_1, x_2, \ldots, x_k\}$ denote a set of distinct coset representatives for the normal subgroup N in G. In order to analyze the spectral expansion of $\mathrm{Cay}(G, A \cup B)$ we consider the basis $\{xn \mid x \in X, n \in N\}$ of $\mathbb{C}[G]$. The element

$u_N = \frac{1}{|N|}\sum_{n\in N} n$ of $\mathbb{C}[G]$ corresponds to the uniform distribution supported on N. It has the following important properties:

1. For all $a \in N$ $M_a(u_N) = u_N$ because $Na = N$ for each $a \in N$.
2. For any $b \in G$ consider the linear map $\sigma_b : \mathbb{C}[G] \to \mathbb{C}[G]$ defined by conjugation: $\sigma_b(\sum_g \alpha_g g) = \sum_g \alpha_g b^{-1}gb$. Since $N \lhd G$ the linear map σ_b is an automorphism of N. It follows that for all $b \in G$ $\sigma_b(u_N) = u_N$.

Now, consider the subspaces U and W of $\mathbb{C}[G]$ defined as follows:

$$U = \left\{ \left(\sum_{x\in X} \alpha_x x \right) u_N \right\}, \quad W = \left\{ \sum_{x\in X} x \left(\sum_{n\in N} \beta_{n,x} n \right) \;\middle|\; \sum_{n} \beta_{n,x} = 0,\ \forall x \in X \right\}$$

It is easy to see that U and W are indeed subspaces of $\mathbb{C}[G]$. Furthermore, we note that every vector in U is orthogonal to every vector in W with respect to the usual dot product, i.e. $U \perp W$. This follows easily from the fact that xu_N is orthogonal to $x\sum_{n\in N}\beta_{n,x}n$ whenever $\sum_{n\in N}\beta_{n,x}n$ is orthogonal to u_N. Note that $\sum_{n\in N}\beta_{n,x}n$ is indeed orthogonal to u_N when $\sum_{n\in N}\beta_{n,x} = 0$. We claim that $\mathbb{C}[G]$ is a direct sum of its subspaces U and W.

Proposition 1. *The group algebra $\mathbb{C}[G]$ has a direct sum decomposition $\mathbb{C}[G] = U + W$.*

We now prove the main result of this section. For a vector v, $\|v\|$ will denote its standard ℓ_2-norm.

Lemma 1. *Let G be any finite group and N be a normal subgroup of G and $\lambda < 1/2$ be any constant. Suppose A is an expanding generator set for N so that $\mathrm{Cay}(N, A)$ is a λ-spectral expander. Furthermore, suppose $B \subseteq G$ such that $\widehat{B} = \{Nx \mid x \in B\}$ is an expanding generator for the quotient group G/N and $\mathrm{Cay}(G/N, \widehat{B})$ is also a λ-spectral expander. Then $A \cup B$ is an expanding generator set for G such that $\mathrm{Cay}(G, A\cup B)$ is a $\frac{(1+\lambda)(\max\{|A|,|B|\})}{|A|+|B|}$-spectral expander.* [1]

Proof. We give the proof for the case when $|A| = |B|$ (the general case is identical). Let $v \in \mathbb{C}[G]$ be any vector such that $v \perp \mathbf{1}$ and M denote the normalized adjacency matrix of the Cayley graph $\mathrm{Cay}(G, A \cup B)$. Our goal is to show that $\|Mv\| \le (1+\lambda)\|v\|/2$. Notice that the adjacency matrix M can be written as $(M_A + M_B)/2$ where $M_A = \sum_{a\in A} M_a/|A|$ and $M_B = \sum_{b\in B} M_b/|B|$.[2]

Claim 1. For any two vectors $u \in U$ and $w \in W$, we have $M_A u \in U$, $M_A w \in W$, $M_B u \in U$, $M_B w \in W$, i.e. U and W are invariant under the transformations M_A and M_B.

[1] Since A and B are multisets, we can ensure $|A|$ and $|B|$ are within a factor of 2 of each other by scaling up the smaller multiset. We can even ensure that A and B are of the same cardinality which is a power of 2.

[2] In the case when $|A| \neq |B|$, the adjacency matrix M will be $\frac{|A|}{|A|+|B|}M_A + \frac{|B|}{|A|+|B|}M_B$.

Proof. Consider vectors of the form $u = xu_N \in U$ and $w = x\sum_{n\in N}\beta_{n,x}n \in W$, where $x \in X$ is arbitrary. By linearity, it suffices to prove for each $a \in A$ and $b \in B$ that $M_a u \in U$, $M_b u \in U$, $M_a w \in W$, and $M_b w \in W$. Notice that $M_a u = xu_N a = xu_N = u$ since $u_N a = u_N$. Furthermore, we can write $M_a w = x\sum_{n\in N}\beta_{n,x}na = x\sum_{n'\in N}\gamma_{n',x}n'$, where $\gamma_{n',x} = \beta_{n,x}$ and $n' = na$. Since $\sum_{n'\in N}\gamma_{n',x} = \sum_{n\in N}\beta_{n,x} = 0$ it follows that $M_a w \in W$. Now, consider $M_b u = ub$. For $x \in X$ and $b \in B$ the element xb can be *uniquely* written as $x_b n_{x,b}$, where $x_b \in X$ and $n_{x,b} \in N$. Hence, $M_b u = xu_N b = xb(b^{-1}u_N b) = x_b n_{x,b}\sigma_b(u_N) = x_b n_{x,b}u_N = x_b u_N \in U$. Finally, $M_b w = x(\sum_{n\in N}\beta_{n,x}n)b = xb(\sum_{n\in N}\beta_{n,x}b^{-1}nb) = x_b n_{x,b}\sum_{n\in N}\beta_{bnb^{-1},x}n = x_b\sum_{n\in N}\gamma_{n,x}n \in W$. Here, we note that $\gamma_{n,x} = \beta_{n',x}$ and $n' = b(n_{x,b}^{-1}n)b^{-1}$. Hence $\sum_{n\in N}\gamma_{n,x} = 0$, which puts $M_b w$ in the subspace W as claimed. □

Claim 2. Let $u \in U$ such that $u \perp \mathbf{1}$ and $w \in W$. Then $\|M_A u\| \le \|u\|$, $\|M_B w\| \le \|w\|$, $\|M_B u\| \le \lambda\|u\|$, and $\|M_A w\| \le \lambda\|w\|$.

Proof. The first two inequalities follow from the fact that M_A and M_B are the normalized adjacency matrices of the Cayley graphs $\mathrm{Cay}(G, A)$ and $\mathrm{Cay}(G, B)$ respectively.

Now we prove the third inequality. Let $u = (\sum_x \alpha_x x)u_N$ be any vector in U such that $u \perp \mathbf{1}$. Then $\sum_{x\in X}\alpha_x = 0$. Now consider the vector $\widehat{u} = \sum_{x\in X}\alpha_x Nx$ in the group algebra $\mathbb{C}[G/N]$. Notice that $\widehat{u} \perp \mathbf{1}$. Let $M_{\widehat{B}}$ denote the normalized adjacency matrix of $\mathrm{Cay}(G/N, \widehat{B})$. Since it is a λ-spectral expander it follows that $\|M_{\widehat{B}}\widehat{u}\| \le \lambda\|\widehat{u}\|$. Writing out $M_{\widehat{B}}\widehat{u}$ we get $M_{\widehat{B}}\widehat{u} = \frac{1}{|B|}\sum_{b\in B}\sum_{x\in X}\alpha_x Nxb = \frac{1}{|B|}\sum_{b\in B}\sum_{x\in X}\alpha_x Nx_b$, because $xb = x_b n_{x,b}$ and $Nxb = Nx_b$ (as N is a normal subgroup). Hence the norm of the vector $\frac{1}{|B|}\sum_{b\in B}\sum_{x\in X}\alpha_x Nx_b$ is bounded by $\lambda\|\widehat{u}\|$. Equivalently, the norm of the vector $\frac{1}{|B|}\sum_{b\in B}\sum_{x\in X}\alpha_x x_b$ is bounded by $\lambda\|\widehat{u}\|$. On the other hand, we have

$$M_B u = \frac{1}{|B|}\sum_b\left(\sum_x \alpha_x x\right)u_N b = \frac{1}{|B|}\sum_b\left(\sum_x \alpha_x xb\right)b^{-1}u_N b$$
$$= \frac{1}{|B|}\left(\sum_b\sum_x \alpha_x x_b n_{x,b}\right)u_N = \frac{1}{|B|}\left(\sum_b\sum_x \alpha_x x_b\right)u_N.$$

For any vector $(\sum_{x\in X}\gamma_x x)u_N \in U$ it is easy to see that its norm can be written as $\|\sum_{x\in X}\gamma_x x\|\|u_N\|$. Since $\|1/|B|\sum_b\sum_x \alpha_x x_b\| \le \lambda\|\sum_{x\in X}\alpha_x x\|$, we have $\|M_B u\| \le \lambda\|u\|$.

We now show the fourth inequality. For each $x \in X$ it is useful to consider the following subspaces of $\mathbb{C}[G]$: $\mathbb{C}[xN] = \{x\sum_{n\in N}\theta_n n \mid \theta_n \in \mathbb{C}\}$. For any distinct $x \ne x' \in X$, since $xN \cap x'N = \emptyset$, vectors in $\mathbb{C}[xN]$ have support disjoint from vectors in $\mathbb{C}[x'N]$. Hence $\mathbb{C}[xN] \perp \mathbb{C}[x'N]$ which implies that the subspaces $\mathbb{C}[xN], x \in X$ are pairwise mutually orthogonal. Furthermore, the matrix M_A maps $\mathbb{C}[xN]$ to $\mathbb{C}[xN]$ for each $x \in X$.

Now, consider any vector $w = \sum_{x\in X}x\left(\sum_n \beta_{n,x}n\right)$ in W. Letting $w_x = x\left(\sum_{n\in N}\beta_{n,x}n\right) \in \mathbb{C}[xN]$ for each $x \in X$ we note that $M_A w_x \in \mathbb{C}[xN]$ for each $x \in X$. Hence, by Pythogoras theorem we have $\|w\|^2 = \sum_{x\in X} \|w_x\|^2$ and $\|M_A w\|^2 = \sum_{x\in X} \|M_A w_x\|^2$. Since $M_A w_x = x M_A \left(\sum_{n\in N} \beta_{n,x} n\right)$, it follows that $\|M_A w_x\| = \|M_A \left(\sum_{n\in N} \beta_{n,x} n\right)\| \leq \lambda \| \sum_{n\in N} \beta_{n,x} n\| = \lambda \|w_x\|$. Putting it together, it follows that $\|M_A w\|^2 \leq \lambda^2 \left(\sum_{x\in X} \|w_x\|^2\right) = \lambda^2 \|w\|^2$. □

We now complete the proof of the lemma. Consider any vector $v \in \mathbb{C}[G]$ such that $v \perp \mathbf{1}$. Let $v = u + w$ where $u \in U$ and $w \in W$. Let $\langle , \rangle$ denote the inner product in $\mathbb{C}[G]$. Since $M = (M_A + M_B)/2$, we have $\|Mv\|^2 = \frac{1}{4}\langle M_A v, M_A v\rangle + \frac{1}{4}\langle M_B v, M_B v\rangle + \frac{1}{2}\langle M_A v, M_B v\rangle$. We consider each of the three summands in the above expression.

Firstly, since $v = u + w$, we can write $\langle M_A v, M_A v\rangle = \langle M_A u, M_A u\rangle + \langle M_A w, M_A w\rangle + 2\langle M_A u, M_A w\rangle$. By Claim 1 and the fact that $U \perp W$, we get $\langle M_A u, M_A w\rangle = 0$. Thus, $\langle M_A v, M_A v\rangle \leq \|u\|^2 + \lambda^2\|w\|^2$. By an identical argument, Claim 1 and Claim 2 imply $\langle M_B v, M_B v\rangle \leq \lambda^2\|u\|^2 + \|w\|^2$. Finally, $\langle M_A v, M_B v\rangle = \langle M_A u, M_B u\rangle + \langle M_A w, M_B w\rangle$. Now, using the Cauchy-Schwarz inequality and Claim 2, we get $\langle M_A v, M_B v\rangle \leq \lambda(\|u\|^2 + \|w\|^2)$. Combining all the inequalities, we get $\|Mv\|^2 \leq \frac{1}{4}\left(1 + 2\lambda + \lambda^2\right)\left(\|u\|^2 + \|w\|^2\right) = \frac{(1+\lambda)^2}{4}\|v\|^2$. Hence, it follows that $\|Mv\| \leq \frac{1+\lambda}{2}\|v\|$. □

The graph $\mathrm{Cay}(G, A \cup B)$ is $\frac{1+\lambda}{2}$-spectral. Using *derandomized squaring* [16], we can compute from $A \cup B$ an expanding generator set S for G of size $O(|A \cup B|)$, such that $\mathrm{Cay}(G, S)$ is λ-spectral. Details are given in the full version [6]. As a consequence, we obtain the following lemma which we will apply repeatedly. For ease of subsequent exposition, we fix $\lambda = 1/4$ in the following lemma.

Lemma 2. *Let G be a finite group and N be a normal subgroup of G such that $N = \langle A\rangle$ and* $\mathrm{Cay}(N, A)$ *is a* $1/4$*-spectral expander. Further, let $B \subseteq G$ and $\widehat{B} = \{Nx \mid x \in B\}$ such that $G/N = \langle \widehat{B}\rangle$ and* $\mathrm{Cay}(G/N, \widehat{B})$ *is a* $1/4$*-spectral expander. Then in time polynomial in $|A| + |B|$, we can construct an expanding generator set S for G, such that $|S| = O(|A| + |B|)$ and* $\mathrm{Cay}(G, S)$ *is a* $1/4$*-spectral expander.*[3]

3 Normal Series and Solvable Permutation Groups

In section 2, we saw how to construct an expanding generator set for a group G from expanding generator sets for some normal subgroup N and the associated quotient group G/N. We apply it to the entire normal series for a *solvable* group G. More precisely, let $G \leq S_n$ such that $G = G_0 \triangleright G_1 \triangleright \cdots \triangleright G_r = \{1\}$ is a *normal series* for G. That means G_i is a normal subgroup of G for each i, and hence G_i is also a normal subgroup of G_j for each $j < i$. Given expanding generator sets for each of the quotient groups G_i/G_{i+1}, we give an efficient construction of an expanding generator set for G.

[3] The lemma holds for any finite group G with the caveat that group operations in G are polynomial-time computable. However, we require the lemma only for quotient groups H/N where $H, N \leq S_n$, and group operations for such H/N are polynomial-time computable.

Lemma 3. *Let $G \leq S_n$ with normal series $\{G_i\}_{i=0}^r$ be as above. Further, for each i let B_i be a generator set for G_i/G_{i+1} such that* $\mathrm{Cay}(G_i/G_{i+1}, B_i)$ *is a* $1/4$-*spectral expander. Let $s = \max_i\{|B_i|\}$. Then in deterministic time polynomial in n and s we can compute a generator set B for G such that* $\mathrm{Cay}(G, B)$ *is a* $1/4$-*spectral expander and $|B| = c^{\log r}s$ for some constant $c > 0$.*

A detailed proof of this lemma is in the full version [6].

Now we apply the above lemma to solvable permutation groups. Let G be any finite solvable group. The *derived series* for G is the following chain of subgroups of G: $G = G_0 \rhd G_1 \rhd \cdots \rhd G_k = \{1\}$ where, for each i, G_{i+1} is the *commutator subgroup* of G_i. That is G_{i+1} is the normal subgroup of G_i generated by all elements of the form $xyx^{-1}y^{-1}$ for $x, y \in G_i$. It turns out that G_{i+1} is the minimal normal subgroup of G_i such that G_i/G_{i+1} is abelian. Furthermore, the derived series is also a *normal series*. It implies that G_i is a normal subgroup of G_j for each $j < i$.

Our algorithm will crucially exploit a property of the derived series of solvable groups $G \leq S_n$: By a theorem of Dixon [9], the length k of the derived series of a solvable subgroup of S_n is bounded by $5 \log_3 n$. Thus, we get the following result as a direct application of Lemma 3:

Lemma 4. *Suppose $G \leq S_n$ is a solvable group with derived series $G = G_0 \rhd G_1 \rhd \cdots \rhd G_k = \{1\}$ such that for each i we have an expanding generator set B_i for the abelian quotient group G_i/G_{i+1} such that* $\mathrm{Cay}(G_i/G_{i+1}, B_i)$ *is a* $1/4$-*spectral expander. Let $s = \max_i\{|B_i|\}$. Then in deterministic time polynomial in n and s we can compute a generator set B for G such that* $\mathrm{Cay}(G, B)$ *is a* $1/4$-*spectral expander and $|B| = 2^{O(\log k)}s = (\log n)^{O(1)}s$.*

Given a solvable permutation group $G \leq S_n$ by a generator set the polynomial-time algorithm for computing an expanding generator set will proceed as follows: in deterministic polynomial time, we first compute generator sets for each subgroup $\{G_i\}_{1 \leq i \leq k}$ in the derived series for G [13]. In order to apply the above lemma it suffices to compute an expanding generator set B_i for G_i/G_{i+1} such that $\mathrm{Cay}(G_i/G_{i+1}, B_i)$ is $1/4$-spectral. We deal with this problem in the next section.

4 Abelian Quotient Groups

In Section 3, we have seen how to construct an expanding generator set for a solvable group G, from expanding generator sets for the quotient groups G_i/G_{i+1} in the normal series for G. We are now left with the problem of computing expanding generator sets for the abelian quotient groups G_i/G_{i+1}. We state a couple of easy lemmas that will allow us to further simplify the problem. For proofs of these lemmas, we refer the reader to an extended version of this paper [6].

Lemma 5. *Let H and N be subgroups of S_n such that N is a normal subgroup of H and H/N is abelian. Let $p_1 < p_2 < \ldots < p_k$ be the set of all primes bounded* *by n and $e = \lceil \log n \rceil$. Then, there is an onto homomorphism ϕ from the product group $\mathbb{Z}_{p_1^e}^n \times \mathbb{Z}_{p_2^e}^n \times \cdots \times \mathbb{Z}_{p_k^e}^n$ to the abelian quotient group H/N.*

From the proof of Lemma 5, it is obvious that ϕ is computable in poly(n) time. Suppose H_1 and H_2 are two finite groups such that $\phi : H_1 \to H_2$ is an onto homomorphism. In the next lemma we show that the ϕ-image of an expanding generator set for H_1, is an expanding generator set for H_2.

Lemma 6. *Suppose H_1 and H_2 are two finite groups such that $\phi : H_1 \to H_2$ is an onto homomorphism. Furthermore, suppose* $\mathrm{Cay}(H_1, S)$ *is a λ-spectral expander. Then* $\mathrm{Cay}(H_2, \phi(S))$ *is also a λ-spectral expander.*

Now, suppose $H, N \leq S_n$ are groups given by their generator sets, where $N \lhd H$ and H/N is abelian. By Lemmas 5 and 6, it suffices to describe a polynomial (in n) time algorithm for computing an expanding generator set of size $\widetilde{O}(n^2)$ for the product group $\mathbb{Z}_{p_1^e}^n \times \mathbb{Z}_{p_2^e}^n \times \cdots \times \mathbb{Z}_{p_k^e}^n$ such that the second largest eigenvalue of the corresponding Cayley graph is bounded by $1/4$. In the following section, we solve this problem.

4.1 Expanding Generator Set for the Product Group

In this section, we give a deterministic polynomial (in n) time construction of an $\widetilde{O}(n^2)$ size expanding generator set for the product group $\mathbb{Z}_{p_1^e}^n \times \ldots \times \mathbb{Z}_{p_k^e}^n$ such that the corresponding Cayley graph is a $1/4$-spectral expander.

Consider the following *normal series* for this product group given by the subgroups $K_i = \mathbb{Z}_{p_1^{e-i}}^n \times \ldots \times \mathbb{Z}_{p_k^{e-i}}^n$ for $0 \leq i \leq e$. Clearly, $K_0 \rhd K_1 \rhd \cdots \rhd K_e = \{1\}$. This is obviously a normal series since $K_0 = \mathbb{Z}_{p_1^e}^n \times \ldots \times \mathbb{Z}_{p_k^e}^n$ is abelian. Furthermore, $K_i/K_{i+1} = \mathbb{Z}_{p_1}^n \times \ldots \times \mathbb{Z}_{p_k}^n$. Since the length of this series is $e = \lceil \log n \rceil$ we can apply Lemma 3 to construct an expanding generator set of size $\widetilde{O}(n^2)$ for K_0 in polynomial time assuming that we can compute an expanding generator set of size $\widetilde{O}(n^2)$ for $\mathbb{Z}_{p_1}^n \times \ldots \times \mathbb{Z}_{p_k}^n$ in deterministic polynomial time. Thus, it suffices to efficiently compute an $\widetilde{O}(n^2)$-size expanding generator set for the product group $\mathbb{Z}_{p_1}^n \times \ldots \times \mathbb{Z}_{p_k}^n$.

In [1], Ajtai et al, using some number theory, gave a deterministic polynomial time expanding generator set construction for the cyclic group $\mathbb{Z}_t$, where t is given in *binary*.

Theorem 1 ([1]). *Let t be a positive integer given in binary as an input. Then there is a deterministic polynomial-time (i.e. in* $\mathrm{poly}(\log t)$ *time) algorithm that computes an expanding generator set T for $\mathbb{Z}_t$ of size* $O(\log^* t \log t)$*, where* $\log^* t$ *is the least positive integer k such that a tower of k 2's bounds t from above. Furthermore,* $\mathrm{Cay}(\mathbb{Z}_t, T)$ *is λ-spectral for any constant λ.*

Now, consider the group $\mathbb{Z}_{p_1 p_2 \ldots p_k}$. Since $p_1 p_2 \ldots p_k$ can be represented by $O(n \log n)$ bits in binary, we apply the above theorem (with $\lambda = 1/4$) to compute an expanding generator set of size $\widetilde{O}(n)$ for $\mathbb{Z}_{p_1 p_2 \ldots p_k}$ in poly(n) time.

Let $m = O(\log n)$ be a positive integer to be fixed in the analysis later. Consider the product group $M_0 = \mathbb{Z}_{p_1}^m \times \mathbb{Z}_{p_2}^m \times \ldots \mathbb{Z}_{p_k}^m$ and for $1 \le i \le m$ let $M_i = \mathbb{Z}_{p_1}^{m-i} \times \mathbb{Z}_{p_2}^{m-i} \times \ldots \times \mathbb{Z}_{p_k}^{m-i}$. Clearly, the groups M_i form a *normal* series for M_0: $M_0 \rhd M_1 \rhd \cdots \rhd M_m = \{1\}$, and the quotient groups are $M_i/M_{i+1} = \mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \ldots \times \mathbb{Z}_{p_k} = \mathbb{Z}_{p_1 p_2 \ldots p_k}$ (recall that p_i's are all distinct). Now we compute (in poly(n) time) an expanding generator set for $\mathbb{Z}_{p_1 p_2 \cdots p_k}$ of size $\widetilde{O}(n)$ using Theorem 1. Then, we apply Lemma 3 to the above normal series and compute an expanding generator set of size $\widetilde{O}(n)$ for the product group M_0 in polynomial time. The corresponding Cayley graph will be a 1/4-spectral expander. Now we are ready to describe the expanding generator set construction for $\mathbb{Z}_{p_1}^n \times \mathbb{Z}_{p_2}^n \times \ldots \times \mathbb{Z}_{p_k}^n$.

The Final Construction. The construction is based on the technique developed in [3]. For $1 \le i \le k$ let m_i be the least positive integer such that $p_i^{m_i} > cn$ (where c is a suitably large constant). Thus, $p_i^{m_i} \le cn^2$ for each i. For each i, $\mathbb{F}_{p_i^{m_i}}$ be the finite field of $p_i^{m_i}$ elements which can be deterministically constructed in polynomial time since it is polynomial sized. Let $m = \max\{m_i\}_{i=1}^k$ which is still of $O(\log n)$. Clearly, there is an onto homomorphism ψ from the group $\mathbb{Z}_{p_1}^m \times \mathbb{Z}_{p_2}^m \times \ldots \times \mathbb{Z}_{p_k}^m$ to the additive group of $\mathbb{F}_{p_1^{m_1}} \times \mathbb{F}_{p_2^{m_2}} \times \ldots \times \mathbb{F}_{p_k^{m_k}}$. Thus, if S is the expanding generator set of size $\widetilde{O}(n)$ constructed above for $\mathbb{Z}_{p_1}^m \times \mathbb{Z}_{p_2}^m \times \ldots \times \mathbb{Z}_{p_k}^m$, it follows from Lemma 6 that $\psi(S)$ is an expanding generator multiset of size $\widetilde{O}(n)$ for the additive group $\mathbb{F}_{p_1^{m_1}} \times \mathbb{F}_{p_2^{m_2}} \times \ldots \times \mathbb{F}_{p_k^{m_k}}$. Define $T \subset \mathbb{F}_{p_1^{m_1}} \times \mathbb{F}_{p_2^{m_2}} \times \ldots \times \mathbb{F}_{p_k^{m_k}}$ to be any (say, the lexicographically first) set of cn many k-tuples such that any two tuples $(x_1, x_2, \ldots, x_k)$ and $(x_1', x_2', \ldots, x_k')$ in T are distinct in all coordinates. Thus $x_j \ne x_j'$ for all $j \in [k]$. It is obvious that we can construct T by picking the first cn such tuples in lexicographic order.

We now define the expanding generator set R. Let $x = (x_1, x_2, \ldots, x_k) \in T$ and $y = (y_1, y_2, \ldots, y_k) \in \psi(S)$. Define $v_i = (\langle 1, y_i \rangle, \langle x_i, y_i \rangle, \ldots, \langle x_i^{n-1}, y_i \rangle)$ where $x_i^j \in \mathbb{F}_{p_i^{m_i}}$ and $\langle x_i^j, y_i \rangle$ is the dot product modulo p_i of the elements x_i^j and y_i seen as p_i-tuples in $\mathbb{Z}_{p_i}^{m_i}$. Hence, v_i is an n-tuple and $v_i \in \mathbb{Z}_{p_i}^n$. Now define $R = \{(v_1, v_2, \ldots, v_k) \mid x \in T, y \in \psi(S)\}$. Notice that $|R| = \widetilde{O}(n^2)$. Using ideas from [3] we can prove that R is an expanding generator set for $\mathbb{Z}_{p_1}^n \times \mathbb{Z}_{p_2}^n \times \ldots \times \mathbb{Z}_{p_k}^n$.

Claim. R is an expanding generator set for the product group $\mathbb{Z}_{p_1}^n \times \ldots \times \mathbb{Z}_{p_k}^n$.

Proof. Let $(\chi_1, \chi_2, \ldots, \chi_k)$ be a nontrivial character of the product group $\mathbb{Z}_{p_1}^n \times \mathbb{Z}_{p_2}^n \times \ldots \times \mathbb{Z}_{p_k}^n$, i.e. there is at least one j such that χ_j is nontrivial. Let ω_i be a primitive p_i^{th} root of unity. Recall that, since χ_i is a character there is a corresponding vector $\beta_i \in \mathbb{Z}_{p_i}^n$, i.e. $\chi_i : \mathbb{Z}_{p_i}^n \to \mathbb{C}$ and $\chi_i(u) = \omega_i^{\langle \beta_i, u \rangle}$ for $u \in \mathbb{Z}_{p_i}^n$ and the inner product in the exponent is a modulo p_i inner product. The character χ_i is nontrivial if and only if β_i is a nonzero element of $\mathbb{Z}_{p_i}^n$.

It is well-known that the characters $(\chi_1, \chi_2, \ldots, \chi_k)$ of the abelian group $\mathbb{Z}_{p_1}^n \times \mathbb{Z}_{p_2}^n \times \ldots \times \mathbb{Z}_{p_k}^n$ are also the eigenvectors for the adjacency matrix of the Cayley graph of the group with any generator set (see Proposition 11.7

of [10]). Thus, in order to prove that R is an expanding generator set for $\mathbb{Z}_{p_1}^n \times \mathbb{Z}_{p_2}^n \times \ldots \times \mathbb{Z}_{p_k}^n$, it is enough to bound the following estimate for the nontrivial characters $(\chi_1, \chi_2, \ldots, \chi_k)$ since that directly bounds the second largest eigenvalue in absolute value.

$$\begin{aligned}\left|\mathbb{E}_{x\in T, y\in\psi(S)}[\chi_1(v_1)\chi_2(v_2)\ldots\chi_k(v)]\right| &= \left|\mathbb{E}_{x\in T, y\in\psi(S)}[\omega_1^{\langle\beta_1, v_1\rangle}\ldots\omega_k^{\langle\beta_k, v_k\rangle}]\right| \\ &= \left|\mathbb{E}_{x,y}[\omega_1^{\langle q_1(x_1), y_1\rangle}\ldots\omega_k^{\langle q_k(x_k), y_k\rangle}]\right| \\ &\le \mathbb{E}_x\left|\mathbb{E}_y[\omega_1^{\langle q_1(x_1), y_1\rangle}\ldots\omega_k^{\langle q_k(x_k), y_k\rangle}]\right|,\end{aligned}$$

where $q_i(x) = \sum_{\ell=0}^{n-1}\beta_{i,\ell}x^\ell \in \mathbb{F}_{p_i}[x]$ for $\beta_i = (\beta_{i,0}, \beta_{i,1}, \ldots, \beta_{i,n-1})$. Since the character is nontrivial, suppose $\beta_j \neq 0$, then q_j is a nonzero polynomial of degree at most $n-1$. Hence the probability that $q_j(x_j) = 0$, when x is picked from T is bounded by $\frac{n}{cn}$. On the other hand, when $q_j(x_j) \neq 0$ the tuple $(q_1(x_1), \ldots, q_k(x_k))$ defines a nontrivial character of the group $\mathbb{Z}_{p_1}^m \times \ldots \times \mathbb{Z}_{p_k}^m$. Since S is an expanding generator set for the abelian group $\mathbb{Z}_{p_1}^m \times \ldots \times \mathbb{Z}_{p_k}^m$, the character defined by $(q_1(x_1), \ldots, q_k(x_k))$ is also an eigenvector for $\mathbb{Z}_{p_1}^m \times \ldots \times \mathbb{Z}_{p_k}^m$, in particular w.r.t. generator set S. Hence, $|\mathbb{E}_{y\in S}[\omega_1^{\langle q_1(x_1), y_1\rangle}\ldots\omega_k^{\langle q_k(x_k), y_k\rangle}]| \le \varepsilon$, where the parameter ε can be fixed to an arbitrary small constant by Theorem 1. Hence the above estimate is bounded by $\frac{n}{cn} + \varepsilon = \frac{1}{c} + \varepsilon$ which can be made $\le 1/4$ by choosing c and ϵ suitably. □

To summarize, Claim 4.1 along with Lemmas 5 and 6 directly yields the following theorem.

Theorem 2. *In deterministic polynomial (in n) time we can construct an expanding generator set of size $\widetilde{O}(n^2)$ for the product group $\mathbb{Z}_{p_1}^n \times \cdots \times \mathbb{Z}_{p_k}^n$ (where for each i, p_i is a prime number $\le n$) that makes it a $1/4$-spectral expander. Consequently, if H and N are subgroups of S_n given by generator sets and H/N is abelian then in deterministic polynomial time we can compute an expanding generator set of size $\widetilde{O}(n^2)$ for H/N that makes it a $1/4$-spectral expander.*

Finally, we state the main theorem which follows directly from the above theorem and Lemma 4.

Theorem 3. *Let $G \le S_n$ be a solvable permutation group given by a generating set. Then in deterministic polynomial time we can compute an expanding generator set S of size $\widetilde{O}(n^2)$ such that the Cayley graph* $\mathrm{Cay}(G, S)$ *is a $1/4$-spectral expander.*

On a related note, in the case of general permutation groups we have the following theorem about computing expanding generator sets. The proof, omitted here, can be found in the full version [6].

Theorem 4. *Given $G \le S_n$ by a generator set S' and $\lambda > 0$, we can deterministically compute (in time* $\mathrm{poly}(n, |S'|)$*) an expanding generator set T for G such that* $\mathrm{Cay}(G, T)$ *is a λ-spectral expander and $|T| = O(n^{16q+10}\left(\frac{1}{\lambda}\right)^{32q})$ (where q is a large constant).*

5 Small Bias Spaces for $\mathbb{Z}_d^n$

We note that the expanding generator set construction for abelian groups in the previous section also gives a new construction of ε-bias spaces for $\mathbb{Z}_d^n$, which we now describe.

In [7] Azar, Motwani, and Naor first considered the construction of ε-bias spaces for abelian groups, specifically for the group $\mathbb{Z}_d^n$. For arbitrary d and any $\varepsilon > 0$ they construct ε-bias spaces of size $O((d + n^2/\varepsilon^2)^C)$, where C is the constant in Linnik's Theorem. The construction involves finding a suitable prime (or prime power) promised by Linnik's theorem which can take time up to $O((d+n^2)^C)$. The current best known bound for C is $\leq 11/2$ (and assuming ERH it is 2). Their construction yields a polynomial-size ε-bias space for $d = n^{O(1)}$.

It is interesting to compare this result of [7] with our results. Let d be any positive integer with prime factorization $p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$. So each p_i is $O(\log d)$ bit sized and each e_i is bounded by $O(\log d)$. Given d as input in unary, we can efficiently find the prime factorization of d. Using the result of Wigderson and Xiao [17], we compute an $O(\log d)$ size expanding generator set for $\mathbb{Z}_{p_1 p_2 \ldots p_k}$ in deterministic time polynomial in d. Then we construct an expanding generator set of size $O(\text{poly}(\log n) \log d)$ for $\mathbb{Z}_{p_1}^m \times \ldots \times \mathbb{Z}_{p_k}^m$ for $m = O(\log n)$ using the method described in Section 4.1. It then follows from Section 4.1 that we can construct an $O(n\,\text{poly}(\log n) \log d)$ size expanding generator set for $\mathbb{Z}_{p_1}^n \times \ldots \times \mathbb{Z}_{p_k}^n$ in deterministic polynomial time. Finally, from Section 4.1, it follows that we can construct an $O(n\,\text{poly}(\log n, \log d))$ size expanding generator set for $\mathbb{Z}_d^n$ (which is isomorphic to $\mathbb{Z}_{p_1^{e_1}}^n \times \ldots \mathbb{Z}_{p_k^{e_k}}^n$) since each e_i is bounded by $\log d$. Now for any arbitrary $\varepsilon > 0$, the explicit dependence of ε in the size of the generator set is $(1/\epsilon)^{32q}$. We summarize the discussion in the following theorem.

Theorem 5. *Let d, n be any positive integers (in unary) and $\varepsilon > 0$. Then, in deterministic* $\text{poly}(n, d, \frac{1}{\varepsilon})$ *time, we can construct an* $O(n\,\text{poly}(\log n, \log d))(1/\varepsilon)^{32q}$ *size ε-bias space for $\mathbb{Z}_d^n$.*

6 Final Remarks

The Alon-Roichman theorem guarantees the existence of $O(n \log n)$ size expanding generator sets for permutation groups $G \leq S_n$. In this paper, we construct $\widetilde{O}(n^2)$ size expanding generator sets for solvable groups. But for non-solvable permutation groups our construction is far from optimal. On the other hand, our construction of ε-bias space for $\mathbb{Z}_d^n$ is very different from the construction of [7] and improves upon it in terms of d and n, although it is worse in terms of the parameter ε. Finding efficient constructions that improve these bounds is an interesting open problem.

Acknowledgements. We are grateful to the anonymous referees for their valuable comments. We thank Shachar Lovett for pointing out the result of Ajtai et al [1]. We also thank Avi Wigderson for his comments and suggestions.

References

1. Ajtai, M., Iwaniec, H., Komlós, J., Pintz, J., Szemerédi, E.: Construction of a thin set with small Fourier coeffciients. Bull. London Math. Soc. 22, 583–590 (1990)
2. Alon, N., Bruck, J., Naor, J., Naor, M., Roth, R.M.: Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. IEEE Transactions on Information Theory 38(2), 509–516 (1992)
3. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple construction of almost k-wise independent random variables. Random Struct. Algorithms 3(3), 289–304 (1992)
4. Alon, N., Roichman, Y.: Random Cayley Graphs and Expanders. Random Struct. Algorithms 5(2), 271–285 (1994)
5. Arvind, V., Mukhopadhyay, P., Nimbhorkar, P.: Erdös-Rényi Sequences and Deterministic construction of Expanding Cayley Graphs. Electronic Colloquium on Computational Complexity (ECCC) 18, 81 (2011)
6. Arvind, V., Mukhopadhyay, P., Nimbhorkar, P., Vasudev, Y.: Expanding generator sets for solvable permutation groups. Electronic Colloquium on Computational Complexity (ECCC) 18, 140 (2011)
7. Azar, Y., Motwani, R., Naor, J.: Approximating Probability Distributions Using Small Sample Spaces. Combinatorica 18(2), 151–171 (1998)
8. Babai, L.: Local expansion of vertex-transitive graphs and random generation in finite groups. In: STOC, pp. 164–174 (1991)
9. Dixon, J.D.: The solvable length of a solvable linear group. Mathematische Zeitschrift 107, 151–158 (1968), doi:10.1007/BF01111027
10. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bull. Amer. Math. Soc. 43, 439–561 (2006)
11. Lubotzky, A., Weiss, B.: Groups and expanders. In: Friedman, J. (ed.) Expanding Graphs. DIMACS Ser. Discrete Math. Theoret. Compt. Sci, vol. 10, pp. 95–109 (1993)
12. Lubotzky, A., Phillips, R., Sarnak, P.: Ramanujan graphs. Combinatorica 8(3), 261–277 (1988)
13. Luks, E.M.: Permutation groups and polynomial-time computation. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 11, pp. 139–175 (1993)
14. Reingold, O.: Undirected connectivity in log-space. J. ACM 55(4), 17:1–17:24 (2008)
15. Reingold, O., Vadhan, S.P., Wigderson, A.: Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In: FOCS, pp. 3–13 (2000)
16. Rozenman, E., Vadhan, S.P.: Derandomized Squaring of Graphs. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 436–447. Springer, Heidelberg (2005)
17. Wigderson, A., Xiao, D.: Derandomizing the Ahlswede-Winter matrix-valued Chernoff bound using pessimistic estimators, and applications. Theory of Computing 4(1), 53–76 (2008)

Generating Functions of Timed Languages[⋆]

Eugene Asarin[1,⋆⋆], Nicolas Basset[2], Aldric Degorre[1], and Dominique Perrin[2]

[1] LIAFA, University Paris Diderot and CNRS, France
[2] LIGM, University Paris-Est Marne-la-Vallée and CNRS, France

Abstract. In order to study precisely the growth of timed languages, we associate to such a language a generating function. These functions (tightly related to volume and entropy of timed languages) satisfy compositionality properties and, for deterministic timed regular languages, can be characterized by integral equations. We provide procedures for closed-form computation of generating functions for some classes of timed automata and regular expressions.

1 Introduction

Since the introduction of timed automata in [1], these automata and their languages are extensively studied both in theoretical perspective and in applications to verification of real-time systems. However, the natural question of measuring the size of timed languages was addressed only recently in [4,3] and a couple of subsequent works. In these articles we explored the asymptotic behavior of the volume of a timed language when the number of events tends to ∞. We showed that for most deterministic timed automata this volume grows (or decreases) exponentially, defined entropy as its growth rate, characterized this entropy as a logarithm of the spectral radius of an integral operator Ψ and showed how to compute the entropy symbolically or numerically.

We believe that size analysis can be useful in several aspects: entropy is a measure of information content in timed words [4] and a key to a timed code theory (work in progress). Whenever the entropy is not too small, timed automata have nice robustness properties [5]. As a practical perspective, we are exploring applications of size analysis to random generation and compression of timed words. We also find the study of the size of timed languages a natural and mathematically appealing generalization of classical results on regular languages and formal series.

In this article, we make a much more precise size analysis of timed languages accepted by deterministic timed automata. We associate to such a language L the sequence of its volumes $\mathrm{Vol}(L_n)$, and the generating function $f(z) = \sum_n \mathrm{Vol}(L_n) z^n$.

[⋆] The support of Agence Nationale de la Recherche under the project EQINOCS (ANR-11-BS02-004) is gratefully acknowledged.

[⋆⋆] A part of the work of this author was done during his stay at the Institute of Mathematical Sciences (National University of Singapore) within its Programme "Automata Theory and Applications".

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 124–135, 2012.

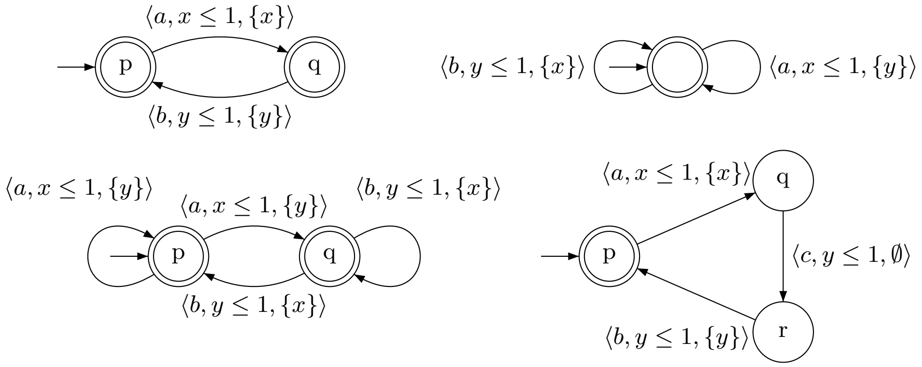

$\langle a, x \leq 1, \{x\}\rangle$
p
q
$\langle b, y \leq 1, \{y\}\rangle$
$\langle b, y \leq 1, \{x\}\rangle$
$\langle a, x \leq 1, \{y\}\rangle$
$\langle a, x \leq 1, \{y\}\rangle$
$\langle a, x \leq 1, \{y\}\rangle$
$\langle b, y \leq 1, \{x\}\rangle$
p
q
$\langle b, y \leq 1, \{x\}\rangle$
$\langle a, x \leq 1, \{x\}\rangle$
q
p
$\langle c, y \leq 1, \emptyset\rangle$
r
$\langle b, y \leq 1, \{y\}\rangle$

Fig. 1. Timed automata. First line: $\mathcal{A}_1$, $\mathcal{A}_2$; second line: $\mathcal{A}_3$, $\mathcal{A}_4$.

Thus the function $f(z)$ contains a complete information on the "size profile" of $\mathrm{Vol}(L_n)$ as a function of n. To relate it to the previous work, we show that $f(z)$ can be expressed in terms of the resolvent of the operator Ψ, and that the entropy of a timed language depends only on the convergence radius of $f(z)$.

Throughout the paper we use examples in Fig. 1, to illustrate the notions of volume, generating function and techniques for computing the latter. Thus, the timed language recognized by automaton $\mathcal{A}_1$ is $L = \{t_1, a, t_2, b, t_3, a, \dots | \forall i \ (t_i + t_{i+1} \leq 1)\}$. For any number of events n we have a polytope in $\mathbb{R}^n$: $L_n = \{t_1, t_2, t_3, \dots, t_n | \forall i \ (t_i + t_{i+1} \leq 1)\}$, the sequence of volumes V_n of these polytopes is $1; 1; \frac{1}{2}; \frac{1}{3}; \frac{5}{24}; \frac{2}{15}; \frac{61}{720}; \frac{17}{315}; \frac{277}{8064} \dots$, and it was shown in [3] that this sequence behaves asymptotically like $(2/\pi)^n$. The methods developed in this paper yield a closed-form expression for the generating function of volumes: $\tan z + \sec z$. The convergence radius of the series, $\pi/2$, is the inverse of the growth rate of the sequence V_n. This series describes precisely the sequence of volumes, and a closed-form formula for V_n can be deduced: $V_{2n-1} = B_{2n}(-4)^n(1-4^n)/(2n)!$; $V_{2n} = (-1)^n E_{2n}/(2n)!$, where B_s stand for Bernoulli numbers and E_s for Euler numbers.

Generating functions behave in a natural way with respect to simple operations on timed languages (disjoint union, unambiguous concatenation, and unambiguous star). However in order to obtain an exact characterization and eventually closed-form expressions for generating function of timed regular languages a more involved analysis is needed. Such an analysis constitutes the main contribution of the article.

Related Work. Our generating functions generalize those of regular languages, thoroughly studied and applied, [6,8,9]. We are not aware of any work on generating functions of timed languages. Techniques and ideas used in this article build on our previous works on volumes and entropy of timed languages; especially on [3] (however the current article is self-contained). As for automata and languages under study, we investigate timed regular languages of [1], clock languages and expressions as in [7], and subclasses of timed automata: regenerating automata from [10] and $1\frac{3}{4}$-clocks automata, that extend both regenerating automata and $1\frac{1}{2}$-clocks automata from [3].

Whenever the alphabet of a timed automaton contains only one letter, its language can be seen as a sequence of polytopes $P_n \subseteq \mathbb{R}^n$. Some of such sequences, known as Fibonacci polytopes, have been studied (independently of timed automata) in combinatorics (see [11] and references therein). In particular, [11] provides a full analysis of the sequence of polytopes produced by automaton $\mathcal{A}_1$ on Fig. 1. Thus our work points at a connection between timed automata and enumerative combinatorics.

Article Structure. In Sect. 2 we introduce a formalism (inspired by [7]) for timed and clock languages, introduce volume functions of such languages, and investigate the properties of these functions. In Sect. 3 we introduce generating functions of timed languages and investigate their general properties. In Sect. 4 we explain how to compute generating functions for several subclasses of timed automata. We summarize the contributions and discuss the directions of future work in Sect. 5.

A longer version [2] of this article with additional proof details and examples is available on-line.

2 Preliminaries

2.1 Clock Languages and Timed Languages

In this paper, we study timed languages (mostly regular) using an approach based on clock languages introduced in [7]. We present this approach in a slightly different form along with a multi-stage semantics. The general idea is as follows: we are interested in timed languages. Timed languages are obtained as projections of clock languages. Clock languages are homomorphic images of discrete "triplet languages". Triplet languages, in turn, can be generated by automata or regular expressions. Below we define formally all these notions and illustrate them on a running example.

An alphabet of *timed events* is the product $\mathbb{R}^+ \times \Sigma$ where Σ is a finite alphabet. The meaning of a timed event (t, a) is that t is the time delay before the event a. A *timed word* is a sequence of timed events and a *timed language* is just a set of timed words.

Inspired by [7], we enrich timed words and languages with d-dimensional clock vectors. A *clock* is a variable which takes values in $\mathbb{R}^+$. In our setting, values of clocks will be bounded by a positive integer M. A *clock word* is a timed word together with an initial and a final clock vector, i.e. an element of $\mathbb{R}^d \times (\mathbb{R}^+ \times \Sigma)^* \times \mathbb{R}^d$. Two clock words $[\mathbf{x}\|\mathbf{w}\|\mathbf{y}]$ and $[\mathbf{x}'\|\mathbf{w}'\|\mathbf{y}']$ are said to be compatible if $\mathbf{y} = \mathbf{x}'$, in this case we define their product by $[\mathbf{x}\|\mathbf{w}\|\mathbf{y}] \cdot [\mathbf{y}\|\mathbf{w}'\|\mathbf{y}'] = [\mathbf{x}\|\mathbf{w}\mathbf{w}'\|\mathbf{y}']$. A *clock language* is a set of clock words. The product of two clock languages $\mathcal{L}$ and $\mathcal{L}'$ is

$$\mathcal{L} \cdot \mathcal{L}' = \{c \cdot c' \mid c \in \mathcal{L},\ c' \in \mathcal{L}',\ c \text{ and } c' \text{ compatible}\}. \tag{1}$$

The neutral element $\mathcal{E}$ is $\{[\mathbf{x}\|\epsilon\|\mathbf{x}] \mid \mathbf{x} \in \mathbb{R}^d\}$ and the Kleene star of a language $\mathcal{L}$ is as usual $\mathcal{L}^* = \bigcup_k \mathcal{L}^k$ with $\mathcal{L}^0 = \mathcal{E}$.

A clock language $\mathcal{L}$ is said to be *deterministic* whenever for each clock word the final clock vector is uniquely determined by the initial clock vector and the timed word, in other words there exists a function $\sigma_{\mathcal{L}} : \mathbb{R}^d \times (\mathbb{R}^+ \times \Sigma)^* \to \mathbb{R}^d$ such that for any clock word $[\mathbf{x}\|\mathbf{w}\|\mathbf{y}]$ of $\mathcal{L}$, we have that $\mathbf{y} = \sigma_{\mathcal{L}}(\mathbf{x}, \mathbf{w})$. In the following, we work with deterministic clock languages[1].

To a clock language we associate its timed projections. Given $\mathcal{L}$, we define $\mathcal{L}(\mathbf{x}, \mathbf{x}')$ as the timed language leading from $\mathbf{x}$ to an element lower than $\mathbf{x}'$: $\mathcal{L}(\mathbf{x}, \mathbf{x}') = \{w \mid \exists \mathbf{y}\, [\mathbf{x}\|\mathbf{w}\|\mathbf{y}] \in \mathcal{L} \wedge \mathbf{y} \leq \mathbf{x}'\}$. We also define the timed language $\mathcal{L}(\mathbf{x}) = \{w \mid \exists \mathbf{y}\, [\mathbf{x}\|\mathbf{w}\|\mathbf{y}] \in \mathcal{L}\}$ as the language starting from $\mathbf{x}$. Note that $\mathcal{L}(\mathbf{x}) = \mathcal{L}(\mathbf{x}, \boldsymbol{M})$ where $\boldsymbol{M} = (M, \ldots, M)$ is the greatest clock vector possible.

2.2 From Timed Automata to Triplet, Clock and Timed Languages

In this section, following [7], we give a convenient representation of timed automata (such as those on Fig. 1) and their languages.

Triplets and Timed Automata. We define *timed automata* as finite automata over the finite alphabet $\mathcal{T}$ of *triplets*. These triplets are tuples $\langle a, \mathfrak{g}, \mathfrak{r}\rangle$ with: a a letter in Σ; $\mathfrak{g}$ a conjunction of constraints, $x_i \bowtie c$ ($i \in \{1..d\}$, $c \in \{0..M\}$, $\bowtie \in \{<, >, \leq, \geq\}$), called *guard*, and $\mathfrak{r} \subseteq \{1..d\}$ a set of indices of clocks to be reset. We suppose moreover that guards are such that all the clocks remain bounded by M.

Clock Semantics of Triplets. Informally, a triplet $\langle a, \mathfrak{g}, \mathfrak{r}\rangle$ corresponds to the following behaviors: starting from some initial clock vector $\mathbf{x}$ let some time t elapse (all the clocks advance by t), check that $\mathfrak{g}$ is satisfied, emit a and update the clocks according to $\mathfrak{r}$. Formally, the clock language of this triplet is $\mathcal{L}(\langle a, \mathfrak{g}, \mathfrak{r}\rangle) = \{[\mathbf{x}\|(t, a)\|\mathfrak{r}(\mathbf{x} + t)] \mid \mathbf{x} + t \models \mathfrak{g}\}$. Here, for a clock vector $\mathbf{x} = (x_1, \ldots, x_d)$, we denote by $\mathbf{x} + t$ the vector $(x_1 + t, \ldots, x_d + t)$. Clock vectors are updated as follows: $\mathfrak{r}(y_1, \ldots, y_d) = (y'_1, \ldots, y'_d)$ with $y'_i = 0$ if $i \in \mathfrak{r}$ and $y'_i = y_i$ otherwise.

This definition can be extended to all triplet words by: $\mathcal{L}(\epsilon) = \mathcal{E}$ and $\mathcal{L}(\pi_1 \ldots \pi_n) = \mathcal{L}(\pi_1) \ldots \mathcal{L}(\pi_n)$ (using the product of clock languages as defined in (1)). Finally for a language $L \subseteq \mathcal{T}^*$, we define $\mathcal{L}(L) = \{\mathcal{L}(\pi) \mid \pi \in L\}$. In fact, $\mathcal{L}$ is a morphism between the two Kleene algebras of triplet and clock languages.

Timed Automata and Their Languages. Given a *timed automaton* $\mathcal{A}$, its *discrete semantics* L is the language of triplet words accepted by $\mathcal{A}$ seen as a finite automaton over $\mathcal{T}$; its *clock semantics* is $\mathcal{L}_{\mathcal{A}} = \mathcal{L}(L)$ and its *timed semantics* is $\mathcal{L}_{\mathcal{A}}(\mathbf{0})$. The timed automata considered in this article are assumed to be deterministic in the sense of [1], i.e. outgoing transitions from the same state with the same letter must have pairwise incompatible guards. Clock languages of deterministic automata are also deterministic.

A timed regular expression is defined as expression over the finite alphabet $\mathcal{T}$. Its discrete, clock and timed semantics are defined similarly to the automata.

[1] Such are clock languages associated to deterministic timed automata. However, a product of two deterministic clock languages can be non-deterministic, and we will explicitly rule out this situation.

This multi-stage timed semantics is equivalent to the usual semantics of timed automata and timed regular expressions.

Example 1 (Our running example). Automata $\mathcal{A}_2$ and $\mathcal{A}_3$ on Fig. 1 have the same discrete semantics[2] which is captured by a regular expression: $(\langle a, x \leq 1, \{y\}\rangle + \langle b, y \leq 1, \{x\}\rangle)^*$. An example of a clock word recognized by the automaton is $[(0.5, 0.8)\|(0.3, a)(0.1, a)(0.9, b)\|(0, 0.9)]$. The timed language recognized is: $\{(t_1, a)\cdots(t_{n_1}, a)(t_{n_1+1}, b)\cdots(t_{n_2}, b)(t_{n_2+1}, a)\cdots(t_{n_3}, a)\cdots \mid \forall j \geq 0,\ \sum_{i=n_j+1}^{n_{j+1}} t_i \leq 1\}$, with $n_0 = 0$ and possibly $n_1 = 0$.

Matrix Notation. A convenient way to see automata is the matrix form. A timed automaton $\mathcal{A}$ over a set of control states Q and an alphabet of transitions $\mathcal{T}$ is uniquely described by three ingredients:

- a $Q \times Q$-matrix $\mathbb{T}$ whose element $\mathbb{T}_{qq'}$ is the set of triplets labelling transitions from q to q';
- a row vector $\mathbf{I}$ describing initial states: for each control state p, its element $I_p = \{\epsilon\}$ iff p is initial, and $\emptyset$ otherwise;
- a column vector $\mathbf{F}$ describing final states: for each control state q, its element $F_q = \{\epsilon\}$ iff q is final, and $\emptyset$ otherwise.

The coefficient $(\mathbb{T}^n)_{p,q}$ of $\mathbb{T}^n$ contains the language of all the triplet words of length n from p to q. The p^{th} coordinates of the column vector $\mathbb{T}^n\mathbf{F}$ contains the language recognized from state p and $\mathbf{I}\mathbb{T}^n\mathbf{F}$ contains the language of triplet words of length n recognized by $\mathcal{A}$. For instance the matrices for $\mathcal{A}_3$ are:

$$\mathbf{I} = \begin{pmatrix} \{\epsilon\} & \emptyset \end{pmatrix};\ \mathbb{T} = \begin{pmatrix} \{\langle a, x \leq 1, \{y\}\rangle\} & \{\langle b, y \leq 1, \{x\}\rangle\} \\ \{\langle a, x \leq 1, \{y\}\rangle\} & \{\langle b, y \leq 1, \{x\}\rangle\} \end{pmatrix};\ \mathbf{F} = \begin{pmatrix} \{\epsilon\} \\ \{\epsilon\} \end{pmatrix}.$$

2.3 Volume(s) of Timed and Clock Languages

Measurable Timed Languages and Clock Languages. A timed language L is *measurable* if, for any word $w \in \Sigma^*$, the projection $L_w = \{\mathbf{t} \in \mathbb{R}^{|w|} \mid (\mathbf{t}, w) \in L\}$[3] is a Lebesgue-measurable subset of $\mathbb{R}^{|w|}$. A clock language $\mathcal{L}$ is measurable if it is deterministic and for every $w \in \Sigma^*$, $\sigma_{\mathcal{L}}(\cdot, (\cdot, w))$ is a Lebesgue-measurable function of $\mathbb{R}^d \times \mathbb{R}^{|w|} \to \mathbb{R}^d$. We remark that timed languages and deterministic clock languages obtained from triplet languages are measurable because their timed projections are polytopes.

Volumes of a Timed Language [3]. The sequence of volumes $(V_n(L))_{n\in\mathbb{N}}$ associated to a measurable timed language is $V_n(L) = \sum_{w\in\Sigma^n} \mathrm{Vol}(L_w)$, where Vol is the hyper-volume (i.e. Lebesgue measure) in $\mathbb{R}^n$. For dimension 0 we define $V_0(L) = 1$ if $\epsilon \in L$, and $V_0(L) = 0$ otherwise.

Now, for a clock language $\mathcal{L}$ and a word $w \in \Sigma^*$ of length $n \geq 0$, we define the clock language $\mathcal{L}(w) = \{[\mathbf{x}\|(\mathbf{t}, v)\|\mathbf{x}'] \mid v = w\}$.

[2] The difference will appear in section 4.1 since $\mathcal{A}_3$ is $1\frac{3}{4}$-clocks and $\mathcal{A}_2$ is not.

[3] by a slight abuse of notation, since $(\mathbb{R} \times \Sigma)^n$ is isomorphic to $\mathbb{R}^n \times \Sigma^n$.

Volumes Constrained by Initial and Final Clock Vectors. Timed regular languages considered below come from clock languages (which themselves come from triplet languages). The information about clock vectors is crucial to compute the volume of timed languages in a compositional manner.

Thus we define parametric volumes depending on initial and final clock vectors as follows $V_n^2(\mathbf{x}, \mathbf{x}') = V_n(\mathcal{L}(\mathbf{x}, \mathbf{x}'))$. We call this function the *cumulative volume function* (CVF)[4] of $\mathcal{L}$. We also allow the following notations: for a clock language $\mathcal{L}$ and a discrete events word w, $V_{\mathcal{L}(w)}^2(\mathbf{x}, \mathbf{x}') = V_{|w|}^2(\mathcal{L}(w)(\mathbf{x}, \mathbf{x}'))$; and for a triplets word π, $V_\pi^2(\mathbf{x}, \mathbf{x}') = V_{|\pi|}(\mathcal{L}(\pi)(\mathbf{x}, \mathbf{x}'))$. The notion of parametric volumes can be also applied to the clock language constrained only by initial clock vector $\mathcal{L}(\mathbf{x})$: $V_n^1(\mathbf{x}) = V_n(\mathcal{L}(\mathbf{x}))$. Clearly $V_n^1(\mathbf{x}) = V_n^2(\mathbf{x}, \infty) = V_n^2(\mathbf{x}, M)$.

CVFs for a Triplet Word. According to the following result, a CVF is easy to compute for a triplet word, and hence for a finite triplet language.

Proposition 1. *For a triplet word π the CVF V_π^2 is piecewise polynomial with rational coefficients of degree $\leq |\pi|$. The pieces are polytopes, and an expression of this function is computable.*

Composing CVFs. In order to define a composition for CVF corresponding to the concatenation of triplet words and languages, we proceed as follows. We define composition of two functions of $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ as: $V_1^2 \star V_2^2(\mathbf{x}, \mathbf{x}') = \int_{\mathbf{y}} V_2^2(\mathbf{y}, \mathbf{x}') V_1^2(\mathbf{x}, d\mathbf{y})$, where the integral is the Lebesgue-Stieltjes integral[5]. We also define $V_1^2 \star v(\mathbf{x}) = \int_{\mathbf{y}} v(\mathbf{y}) V_1^2(\mathbf{x}, d\mathbf{y})$, when v is defined on $\mathbb{R}^d$. Then we can state the key lemma (to transpose concatenation of words to the CVFs world):

Proposition 2. *For any measurable clock languages $\mathcal{L}_1$ and $\mathcal{L}_2$ and discrete words w_1 and w_2, $V_{\mathcal{L}_1(w_1)}^2 \star V_{\mathcal{L}_2(w_2)}^2$ is well defined and satisfies: $V_{\mathcal{L}_1(w_1)}^2 \star V_{\mathcal{L}_2(w_2)}^2 = V_{\mathcal{L}_1(w_1) \cdot \mathcal{L}_2(w_2)}^2$.*

Volume Functions in Timed Automata. As we did for languages, we introduce a Q-vector $\mathbf{V}_n(\mathbf{x})$ of volumes of clock languages and a $Q \times Q$-matrix $\mathbb{V}(\mathbf{x}, \mathbf{x}')$ of cumulative volume functions of elements of the transition matrix $\mathbb{T}$: formally $\mathbf{V}_{n,q}(\mathbf{x}) = V_n(\mathcal{L}_q(\mathbf{x}))$ and $\mathbb{V}_{qq'}(\mathbf{x}, \mathbf{x}') = V_1(\mathcal{L}(\mathbb{T}_{qq'})(\mathbf{x}, \mathbf{x}'))$, with $\mathcal{L}_q = \mathcal{L}((\mathbb{T}^n \mathbf{F})_q)$. It follows from the proposition above that the matrix element $(\mathbb{V}^{\star n})_{pq}$ (of the matrix power wrt $\star$) contains the CVF of $\mathcal{L}((\mathbb{T}^n)_{pq})$, that is of the language of all the clock words of length n leading from p to q. Finally, we get the formula for volumes:

$$\mathbf{V}_n = \mathbb{V}^{\star n} \star \mathbf{V}_F, \tag{2}$$

with $\mathbf{V}_F$ a column vector with $\mathbf{V}_{F,p} = 1$ if p is final, and $\mathbf{V}_{F,p} = 0$ otherwise.

The following not so obvious property of volume functions will be used in the sequel.

Proposition 3. *In a timed automaton $\mathcal{A}$, for $n \geq 1$, the volume functions $\mathbf{V}_n(\mathbf{x})$ and $\mathbb{V}^{\star n}(\mathbf{x}, \mathbf{x}')$ are continuous wrt the initial clock vector $\mathbf{x}$.*

[4] similarly to cumulative distribution functions in probability theory.

[5] By definition, the Lebesgue-Stieltjes integral $\int f(\mathbf{x}) g(d\mathbf{x})$ is the Lebesgue integral of f wrt the measure μ having cumulative distribution function g.

3 Generating Functions

3.1 Definitions

To study volume sequences associated to timed and clock languages we define their generating functions. As usual for generating functions, they allow recovering the sequence, its growth rate, momenta etc; and they have nice compositional properties. Given a timed language L its *generating function* is defined as follows: $f_L(z) = \sum_k z^k V_k(L)$. Given a clock language $\mathcal{L}$, we define a *(parametric) generating function* with a given initial clock vector $f^1_{\mathcal{L}}(z, \mathbf{x}) = \sum_k z^k V_k(\mathcal{L}(\mathbf{x})) = f_L(z)$, with $L = \mathcal{L}(\mathbf{x})$. For a clock language $\mathcal{L}$ we also define another *cumulative generating function* with a given initial clock vector and a bound on the final clock vector: $f^2(z, \mathbf{x}, \mathbf{x}') = \sum_k z^k V^2_k(\mathbf{x}, \mathbf{x}') = f_L(z)$, with $L = \mathcal{L}(\mathbf{x}, \mathbf{x}')$. To summarize, we are interested in computing $f(z)$, but this computation will be based on $f^1(z, \mathbf{x})$, and sometimes on $f^2(z, \mathbf{x}, \mathbf{x}')$.

Given a timed automaton, timed and clock languages, and thus generating functions are naturally associated to its states, for example $f^1_q(z, \mathbf{x}) = \sum_k z^k V_k(\mathcal{L}_q(\mathbf{x})) = f_L(z)$, with $L = \mathcal{L}_q(\mathbf{x})$. Taken for all states, functions f_q and f^1_q form $|Q|$-dimensional vector functions $\mathbf{f}(z, \mathbf{x}), \mathbf{f}^1(z, \mathbf{x})$, while functions $f^2_{q,q'}$ form a $Q \times Q$-matrix function $\mathbb{f}^2(z, \mathbf{x}, \mathbf{x}')$.

3.2 Analytic Characterization

Elementary Properties. First, let us state the relations between the three kinds of generating functions:

Proposition 4. *The functions f, f^1, f^2 are related as follows: $f(z) = f^1(z, \mathbf{0})$; $f^1(z, \mathbf{x}) = f^2(z, \mathbf{x}, \boldsymbol{M})$.*

By definition, f^2, f^1 and f are analytic functions of z. Since we consider timed automata with guards bounded by some constant M, all the volumes V_k (with any initial or final conditions) can be upper bounded by $(M|\Sigma|)^k$. This implies that convergence radius of series for f^2, f^1 and f is at least $(M|\Sigma|)^{-1} > 0$. More precisely, the radius of convergence of f is $1/\limsup_{k\to\infty}(V_k(L))^{1/k} = 2^{-\mathcal{H}(L)}$, where $\mathcal{H}(L)$ is called the volumetric entropy of L (see [3]).

For generating functions associated to timed automata, the following result is a straightforward corollary of Prop. 3:

Proposition 5. *Within its convergence radius, the generating function $\mathbf{f}^1(z, \mathbf{x})$ associated to a timed automaton $\mathcal{A}$ is continuous wrt the initial clock vector $\mathbf{x}$.*

Integral Equation for Generating Functions. Consider a timed automaton. Using formula (2), its generating function can be computed as follows: $\mathbf{f}^1(z, \mathbf{x}) = \sum_k z^k \mathbf{V}_k(\mathbf{x}) = \sum_k z^k \mathbb{V}^{\star k} \star \mathbf{V}_F$, which implies our first main result.

Theorem 1 (Integral equation). *In the interior of its convergence circle, the generating function $\mathbf{f}^1$ is the unique solution of the integral equation*

$$\mathbf{f}^1 - z\mathbb{V} \star \mathbf{f}^1 = \mathbf{V}_F. \tag{3}$$

Example (1, continued). For the automaton $\mathcal{A}_3$ (using the notation $x \dot{-} y$ for $\max(x-y,0)$):

$$\mathbb{V} = \begin{pmatrix} \min(x',1) \dot{-} x \mathbb{1}_{y'\geq 0} & \min(y',1) \dot{-} y \mathbb{1}_{x'\geq 0} \\ \min(x',1) \dot{-} x \mathbb{1}_{y'\geq 0} & \min(y',1) \dot{-} y \mathbb{1}_{x'\geq 0} \end{pmatrix}; \; \mathbf{V}_F = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Equation (3) gives: $f_p^1(z,x,y) = f_q^1(z,x,y) = 1 + z\int_x^1 f_p^1(z,x',0)dx' + z\int_y^1 f_q^1(z,0,y')dy'$. In Section 4.1 below we develop a technique for solving such equations (for a subclass of automata including this one), and compute the generating function for this language.

3.3 Volumes, Generating Functions and Functional Analysis

In this section (which can be skipped by a reader not interested in functional analysis), similarly to [3], we rephrase previous results in terms of the spectral theory of linear operators.

Given a timed automaton, consider the Banach space $\mathcal{F}$ of Q-vectors of continuous functions on clock valuations. Thus an element of $\mathcal{F}$ is a vector $\mathbf{v}$ whose components are continuous $v_q : [0;M]^d \to \mathbb{R}$, and $\mathcal{F} = C([0;M]^d)^Q$. The matrix $\mathbb{V}$ corresponds to an operator $\Psi : \mathcal{F} \to \mathcal{F}$ defined by $\Psi(\mathbf{v}) = \mathbb{V} \star \mathbf{v}$ (a variant of this operator plays the central role in [3]). In terms of this operator, Prop. 3 and equation (2) can be rephrased as follows:

Proposition 6 ([3]). *Ψ is a bounded linear operator on $\mathcal{F}$ (represented by a matrix of integral operators). The volume vector can be obtained by iteration of this operator: $\mathbf{V}_n = \Psi^n(\mathbf{V}_F)$.*

Recall that, by definition, the *resolvent* of an operator A is $R(\lambda, A) = (A - \lambda I)^{-1}$; it is well defined when λ does not belong to the spectrum of A, in particular for $|\lambda| > \rho(A)$, where ρ denotes the spectral radius. We obtain as a consequence of Thm. 1 another characterization of the generating function:

Proposition 7 (Generating function and resolvent). *The generating function $\mathbf{f}^1$ satisfies the formula: $\mathbf{f}^1 = -z^{-1}R(z^{-1}, \Psi)\mathbf{V}_F$, which holds in the interior of the circle $|z| < \rho(\Psi)^{-1}$.*

3.4 Inductive Characterization of Generating Functions

The form of generating functions of finite triplet languages follows from Prop. 1:

Proposition 8. *For a finite triplet language L with maximal word length ℓ, the generating functions f^2, f^1 are piecewise polynomial in $z, \mathbf{x}, \mathbf{x}'$ (pieces are polytopes in $\mathbf{x}, \mathbf{x}'$) of degree $\leq \ell$ wrt z and wrt $\mathbf{x}$ and $\mathbf{x}'$.*

More complex languages can be obtained from finite ones using Kleene algebra operations. As usual in the context of generating functions, we suppose that the operations are unambiguous. A language operation is *ambiguous* if a word of the resulting language can be obtained in several ways by composing different words from the operands. We consider first the simple case of timed languages.

Proposition 9. *Generating functions behave well for unambiguous operations on measurable timed languages:* $f_{L_1\cup L_2}=f_{L_1}+f_{L_2}$; $f_{L_1\cdot L_2}=f_{L_1}f_{L_2}$; $f_{L^*}=1+f_Lf_{L^*}$ *provided* $\epsilon\notin L$.

However, in order to obtain general timed regular languages we need operations on clock languages, which are more involved.

Proposition 10. *Generating functions* f^2 *behave well for unambiguous operations on deterministic measurable clock languages (whenever the resulting language is also deterministic):* $f^2_{\mathcal{L}_1\cup\mathcal{L}_2}=f^2_{\mathcal{L}_1}+f^2_{\mathcal{L}_2}$; $f^2_{\mathcal{L}_1\cdot\mathcal{L}_2}=f^2_{\mathcal{L}_1}\star f^2_{\mathcal{L}_2}$; $f^2_{\mathcal{L}^*}=1\!\!1_{\mathbf{x}\leq\mathbf{x}'}+f^2_{\mathcal{L}}\star f^2_{\mathcal{L}^*}$ *provided* $\mathcal{E}\cap\mathcal{L}=\emptyset$.

Corollary 1. *Generating function* f^1 *for unambiguous compositions of clock languages (under the same hypotheses) can be computed as follows:* $f^1_{\mathcal{L}_1+\mathcal{L}_2}=f^1_{\mathcal{L}_1}+f^1_{\mathcal{L}_2}$; $f^1_{\mathcal{L}_1\cdot\mathcal{L}_2}=f^2_{\mathcal{L}_1}\star f^1_{\mathcal{L}_2}$; $f^1_{\mathcal{L}^*}=1+f^2_{\mathcal{L}}\star f^1_{\mathcal{L}^*}$ *provided* $\mathcal{E}\cap\mathcal{L}=\emptyset$.

4 Computing Generating Functions

The generating function of a timed language represented by an automaton is characterized by a system of integral equations ([3]). The generating function of a timed language represented by a regular expression can be found recursively from piecewise polynomial functions using operations $+,\star$ and solving fixpoint integral equations of Prop. 10 and Cor. 1. Unfortunately, both procedures involve computation of integrals, and solution of integral equations, for this reason, the result cannot be always presented by an explicit formula. Below we consider several subclasses of timed automata, for which generating functions can be obtained in closed form, or at least admit a simpler characterization.

4.1 Generating Functions for Particular Classes of Automata

System of Equations. Our closed-form solutions for subclasses of timed automata will be obtained using a variant of language equations.

Let $Q=G\cup B$ be a disjoint partition of the states of a timed automaton $\mathcal{A}$ into *good* and *bad*. We want to describe the vector $\overline{\mathbf{L}}$ of triplet languages L_q recognized from good states $q\in G$ only. This vector satisfies the equation:

$$\overline{\mathbf{L}}=\overline{\mathbb{T}}\cdot\overline{\mathbf{L}}+\overline{\mathbf{F}},\tag{4}$$

where $\overline{\mathbb{T}}$ is a $G\times G$-matrix and $\overline{\mathbf{F}}$ is a G-vector of triplet languages. Their elements are defined as follows: $\overline{T}_{pq}$ consists of all words leading from p to q via bad states only; $\overline{F}_p$ consists of all words leading from p to a final state via bad states only[6].

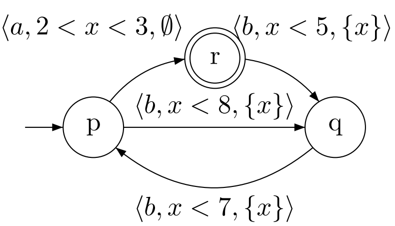


Fig. 2. A regenerating automaton

[6] If this final state is good the word should be ϵ.

Automata with Regeneration. Following [10], we call an automaton *regenerating* if there exists a partition $Q = G \cup B$ having two properties: (a) every cycle in the automaton contains a state in G (good); (b) all the transitions coming into a good state reset all clocks.

W.l.o.g. we suppose that the initial state is good (this can be achieved by adding a new initial state). Condition (a) implies that no cycle is possible within bad states, and thus all the elements of $\overline{\mathbb{T}}$ and $\overline{\mathbf{F}}$ are finite triplet languages (with maximal word length $\leq |B| + 1$). Condition (b) means that (4) can be rewritten in timed languages (instead of clock languages), since when entering in a good state all clocks are reset. This gives

$$\overline{\mathbf{L}}_{\text{timed}} = \overline{\mathbb{T}}_{\text{timed}} \cdot \overline{\mathbf{L}}_{\text{timed}} + \overline{\mathbf{F}}_{\text{timed}}. \tag{5}$$

Applying simple compositionality conditions for generating functions for timed languages (Prop. 9) we obtain that $\overline{\mathbf{f}} = \overline{\mathbb{f}}\ \overline{\mathbf{f}} + \overline{\mathbf{f}}_F$. Due to Prop. 8 all the coefficients (elements of matrix $\overline{\mathbb{f}}$ and vector $\overline{\mathbf{f}}_F$) are polynomials of z. Solving this linear $|G|$-dimensional system we express $\overline{\mathbf{f}}$ as a vector of rational functions of z: $\overline{\mathbf{f}} = \left(I - \overline{\mathbb{f}}\right)^{-1} \overline{\mathbf{f}}_F$. The generating function f of the timed language accepted by the automaton is just one element of this vector $\overline{\mathbf{f}}$. We conclude.

Theorem 2. *For a regenerating automaton the generating function $f(z)$ is a rational function.*

Example 2. Consider a regenerating automaton on Fig. 2. We choose good and bad states as follows: $G = \{p, q\}; B = \{r\}$. The system of equations on timed languages of good states takes the form

$$\begin{pmatrix} L_p \\ L_q \end{pmatrix} = \begin{pmatrix} \emptyset & T_{pq} \\ T_{qp} & \emptyset \end{pmatrix} \cdot \begin{pmatrix} L_p \\ L_q \end{pmatrix} + \begin{pmatrix} F_p \\ \emptyset \end{pmatrix} \text{ with} \tag{6}$$

$$\begin{aligned} T_{pq} =& \{(t_1, a)(t_2, b) | 2 < t_1 < 3 \wedge t_1 + t_2 < 5\} \cup \{(t, b) | t < 8\}; \\ T_{qp} =& \{(t, b) | t < 7\}; \\ F_p =& \{(t, a) | 2 < t < 3\}. \end{aligned}$$

For generating functions this yields: $\begin{pmatrix} f_p \\ f_q \end{pmatrix} = \begin{pmatrix} 0 & 2.5z^2 + 8z \\ 7z & 0 \end{pmatrix} \cdot \begin{pmatrix} f_p \\ f_q \end{pmatrix} + \begin{pmatrix} z \\ 0 \end{pmatrix}$.
Solving this linear system we find the required $f_p(z) = 2z / \left(2 - 35z^3 - 112z^2\right)$. It converges for $|z| < 0.1309$, its Taylor coefficients (i.e. volumes V_n for $n = 0..11$) are $0; 1; 0; 56; 17\frac{1}{2}; 3136; 1960; 175922\frac{1}{4}; 164640; 9885946; 12298479\frac{3}{8}; 556494176$.

$1\frac{3}{4}$-Clocks Automata. We call an automaton $1\frac{3}{4}$-*clocks* if there exists a partition of $Q = G \cup B$ into good and bad states having three properties:(a) every cycle in the automaton contains a good state; (b) the initial state is a good one; (c) for each good state p there is at most one clock $x_{i(p)}$ not reset by incoming transitions.

Similarly to regenerating automata, we apply equations (4), and observe that all the coefficients are finite triplet languages. Unfortunately, since some clocks are not reset, we cannot write an equation on timed languages similar to (5). Instead, we pass to clock languages and their generating functions, as in the general case. This gives:

$$\overline{\mathbf{f}^1} = \overline{\mathbb{f}^2} \star \overline{\mathbf{f}^1} + \overline{\mathbf{f}^1}_F, \tag{7}$$

an integral equation with piecewise polynomial coefficients. We notice that functions in the last equation depend on the clock vector $\mathbf{x} \in \mathbb{R}^d$ (or on two clock vectors $\mathbf{x}$,$\mathbf{x}'$), but in fact for any good state $p \in G$ only one clock $x_{i(p)}$ matters. This allows extracting simpler integral equations from (7), involving only functions of scalar argument.

We proceed as follows: given a G-vector $\mathbf{v}$ whose elements v_p are functions on $\mathbb{R}^d$, we define reduced functions on $\mathbb{R}$: $\widetilde{v}_p(x) = v_p(0, \ldots, 0, x, 0, \ldots, 0)$, with the argument x at position $i(p)$. Reduced G-vector $\widetilde{\mathbf{v}}$ consists of reduced elements $\widetilde{v}_p$. Reduced versions of matrices are defined similarly.

The following identity is based on the requirement of clock resets:

Lemma 1. *For a $1\frac{3}{4}$-clocks automaton the following holds:* $\widetilde{\overline{\mathbb{f}^2} \star \overline{\mathbf{f}^1}} = \widetilde{\overline{\mathbb{f}^2}} \star \widetilde{\overline{\mathbf{f}^1}}$.

Equation (7), reduced to $\widetilde{\overline{\mathbf{f}^1}} = \widetilde{\overline{\mathbb{f}^2}} \star \widetilde{\overline{\mathbf{f}^1}} + \widetilde{\overline{\mathbf{f}^1}_F}$, implies that the reduced vector of generating functions is a solution of equations of the form:

$$\mathbf{f}(z, x) = (\mathbb{A} \star \mathbf{f})(z, x) + \mathbf{b}(z, x), \tag{8}$$

where all the coefficients are piecewise polynomial functions of z and a scalar argument x.

Lemma 2. *An integral equation of the form (8) can be transformed into a system of linear ordinary differential equation with piecewise polynomial coefficients (depending on x and z).*

Theorem 3. *For a $1\frac{3}{4}$-clocks automaton the generating function f can be obtained by solving a system of linear ordinary differential equations with piecewise polynomial coefficients.*

We notice that the theorem gives a rather explicit characterization of f, but not always a closed-form expression.

Example (1, completed). $\mathcal{A}_3$ is $1\frac{3}{4}$-clocks[7] with good states $G = \{p, q\}$ and no bad state $B = \emptyset$. The matrix $\mathbb{A}$ and the vector $\mathbf{b}$ are

$$\mathbb{A} = \begin{pmatrix} z(\min(x', 1) \dot{-} x) & z \min(x', 1) \\ z \min(x', 1) & z(\min(x', 1) \dot{-} x) \end{pmatrix}; \ \mathbf{b} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

We use equation (8) and remark that by symmetry of $\mathbb{A}$, the two generating functions $\widetilde{f^1_p}$ and $\widetilde{f^1_q}$ are equal to a unique function f^1 which satisfies $f^1(z, x) = z \int_x^1 f^1(z, x')\,dx' + z \int_0^1 f^1(z, x')\,dx' + 1$. Differentiating it one time w.r.t x we obtain: $\frac{\partial f^1}{\partial x}(z, x) = -z f^1(z, x)$. The solution has the form $f^1(z, x) = A(z)e^{-zx}$. Remark that $f^1(z, 0) - 1 = 2z \int_0^1 f^1(z, x')\,dx' = 2(f^1(z, 1) - 1)$. We are done since $f(z) = f^1(z, 0) = A(z) = 1/(2e^{-z} - 1)$.

[7] $\mathcal{A}_3$ can be seen as $\mathcal{A}_2$ whose state is split to make it a $1\frac{3}{4}$-clock automaton.

Example 3. Automata $\mathcal{A}_1$ and $\mathcal{A}_4$ are also $1\frac{3}{4}$-clocks, and their generating functions are $\tan z + \sec z$ and $4\big/\big(\pi\left(\mathrm{Bi}'(0)\mathrm{Ai}(-z) - \mathrm{Ai}'(0)\mathrm{Bi}(-z)\right)\big)$, where Ai and Bi stand for Airy functions.

5 Conclusions

In this article, we have introduced generating functions of timed languages, explored their properties and characterized them by integral equations. For subclasses of timed regular languages we have presented closed-form expressions or simpler characterization of generating functions. Generating functions describe with a high precision the quantitative behavior of timed languages.

At the current stage of research, the computation of generating functions is a semi-manual task and restrictions are imposed to the automata. We are planning to explore theoretical and practical algorithmics of timed generating functions, and to implement the algorithm. On the other hand, we want to see whether closed form solutions are possible beyond the class of $1\frac{3}{4}$-clocks languages.

We hope that this approach will lead to new combinatorial results for timed regular languages and sequences of polytopes, better quantitative characterization of such languages with applications to information theory and verification of real-time systems. Also, the approach can be extended to timed formal series, non-regular timed languages, or to richer models such as hybrid automata.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
2. Asarin, E., Basset, N., Degorre, A., Perrin, D.: Generating functions of timed languages. Tech. rep (2012), `http://hal.archives-ouvertes.fr/hal-00678443`
3. Asarin, E., Degorre, A.: Volume and Entropy of Regular Timed Languages: Analytic Approach. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 13–27. Springer, Heidelberg (2009)
4. Asarin, E., Degorre, A.: Volume and Entropy of Regular Timed Languages: Discretization Approach. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 69–83. Springer, Heidelberg (2009)
5. Basset, N., Asarin, E.: Thin and Thick Timed Regular Languages. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 113–128. Springer, Heidelberg (2011)
6. Berstel, J., Reutenauer, C.: Noncommutative rational series with applications. Enc. of Math. and Appl., vol. 137. Cambridge University Press (2011)
7. Bouyer, P., Petit, A.: A Kleene/Büchi-like theorem for clock languages. Journal of Automata, Languages and Combinatorics 7(2), 167–186 (2002)
8. Flajolet, P., Sedgewick, R.: Analytic combinatorics. Cambridge University Press (2009)
9. Salomaa, A., Soittola, M.: Automata-theoretic aspects of formal power series. Springer (1978)
10. Sassoli, L., Vicario, E.: Close form derivation of state-density functions over DBM domains in the analysis of non-Markovian models. In: QEST 2007, pp. 59–68. IEEE Computer Society (2007)
11. Stanley, R.P.: A survey of alternating permutations. In: Combinatorics and graphs. Contemp. Math., vol. 531, pp. 165–196. Amer. Math. Soc., Providence (2010)

The Robust Set Problem: Parameterized Complexity and Approximation

Cristina Bazgan[1,2] and Morgan Chopin[1]

[1] Université Paris-Dauphine, LAMSADE, France
{bazgan,chopin}@lamsade.dauphine.fr
[2] Institut Universitaire de France

Abstract. In this paper, we introduce the k-Robust Set problem: given a graph $G = (V, E)$, a threshold function $t : V \to N$ and an integer k, find a subset of vertices $V' \subseteq V$ of size at least k such that every vertex v in G has less than $t(v)$ neighbors in V'. This problem occurs in the context of the spread of undesirable agents through a network (virus, ideas, fire, ...). Informally speaking, the problem asks to find the largest subset of vertices with the property that if anything bad happens in it then this will have no consequences on the remaining graph. The threshold $t(v)$ of a vertex v represents its reliability regarding its neighborhood; that is, how many neighbors can be infected before v gets himself infected.

We study in this paper the parameterized complexity of k-Robust Set and the approximation of the associated maximization problem. When the parameter is k, we show that this problem is W[2]-complete in general and W[1]-complete if all thresholds are constant bounded. Moreover, we prove that, if $P \neq NP$, the maximization version is not $n^{1-\epsilon}$- approximable for any $\epsilon > 0$ even when all thresholds are at most two. When each threshold is equal to the degree of the vertex, we show that k-Robust Set is fixed-parameter tractable for parameter k and the maximization version is APX-complete. We give a polynomial-time algorithm for graphs of bounded treewidth and a PTAS for planar graphs. Finally, we show that the parametric dual problem $(n - k)$-Robust Set is fixed-parameter tractable for a large family of threshold functions.

1 Introduction

The subject of optimization problems that involve a diffusion process through a network is a large and well-studied topic [15,8,1,12,3]. Such problems share a common idea of selecting an initial subset of vertices to activate in a graph such that, according to a propagation rule, all vertices are activated once the propagation process stops. One such representative problem is the Target Set Selection problem first introduced in [8]: given a graph $G = (V, E)$ and a threshold function $t : V \to N$, the problem asks to find the minimum number of vertices to activate such that all vertices are activated at the end of the propagation process. A vertex v is activated if and only if the number of its activated neighbors is above the *threshold* $t(v)$. This problem has been proved

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 136–147, 2012.

NP-complete even when all thresholds are at most two [8]. Moreover, the problem was surprisingly shown to be hard to approximate within a ratio $O(2^{\log^{1-\epsilon} n})$ for any $\epsilon > 0$ even when all thresholds are at most two [8]. The same inapproximability result holds with *majority thresholds i.e,* for each $v \in V, t(v) = \lceil \frac{d(v)}{2} \rceil$ [8]. From a parameterized perspective, Target Set Selection is W[2]-hard with respect to the solution size for majority thresholds and for thresholds at most two [17]. Furthermore, the problem is proved to be in XP and W[1]-hard with respect to the parameter treewidth [3]. These negative results emphasize the strong intractability nature of the problem.

With regards to the motivation for this study, it is an interesting question to ask the complexity of the converse problem: find the largest subset of vertices such that if a set of vertices gets infected in it then there will be no consequence for all the other vertices. In this context, there is no propagation process involved like in the previous problem; here we want to prevent such phenomenon. This idea of "controlling" the diffusion of dangerous ideas or epidemics is also well-studied [16,13,7]. More formally, we introduce the k-Robust Set problem: given a graph $G = (V, E)$, a threshold function $t : V \rightarrow N$ and an integer k, find a subset of vertices $V' \subseteq V$ of size at least k such that every vertex v in G has less than $t(v)$ neighbors in V'. The set V' is said to be *robust*. Indeed, if one infects any subset $S \subseteq V'$ then there will be no propogation at all since every vertex has a number of infected neighbors below its threshold. Finally, it is worth pointing out that our problem can also be related to a recent paper about the (σ, ρ)-Dominating Set problem [14].

In this paper, we study the parameterized complexity of k-Robust Set and the approximation of the associated maximization problem Max Robust Set. The paper is organized as follows. In Section 2 we give the definitions, terminology and preliminaries. In Section 3 we establish parameterized intractability results for k-Robust Set with various threshold functions. We show that the parametric dual problem $(n-k)$-Robust Set is fixed-parameter tractable for a large family of threshold functions. In Section 4 we give a polynomial-time algorithm to solve k-Robust Set for graphs of bounded treewidth. In Section 5 we establish that Max Robust Set is not $n^{1-\epsilon}$-approximable for any $\epsilon > 0$ even when all thresholds are at most two. If each threshold is equal to the degree of the vertex, we show that Max Robust Set is APX-complete. Conclusion and open problems are given in Section 6. Due to the space limit, some proofs are omitted.

2 Preliminaries

In this section, we give the notation used throughout this paper as well as the statement of the problems. We will conclude this section by providing the basic backgrounds on parameterized complexity and approximation.

Graph Terminology. Let $G = (V, E)$ be an *undirected graph*. The *open neighborhood* of a node $v \in V$, denoted by $N(v)$, is the set of all neighbors of v. The *closed neighborhood* of a node v, denoted $N[v]$, is the set $N(v) \cup \{v\}$. The *degree* of a node v is denoted by $d_G(v)$ or simply $d(v)$ if the graph is clear from context. Let $X \subseteq V$, we denote by $G[X]$ the subgraph of G induced by X.

Problem Definitions. Let $G = (V, E)$ be an undirected graph, and $t : V \to N$ a threshold function. A subset $V' \subseteq V$ is called *robust* if $\forall v \in V, d_{V'}(v) < t(v)$. We call a vertex *bounded* if $t(v) = O(1)$; otherwise, it is said *unbounded.* We define in the following the problems we study in this paper.

k-Robust Set
Input: A graph $G = (V, E)$, a threshold function $t : V \to N$ where $1 \leq t(v) \leq d(v)$ for every $v \in V$, and an integer k
Parameter: k
Question: Is there a robust set $V' \subseteq V$ of size at least k?

We considered also the parametric dual problem $(n - k)$-Robust Set which asks for the existence of a robust set of size at least $n - k$.

The optimization version of k-Robust Set is defined as follows.

Max Robust Set
Input: A graph $G = (V, E)$ and a threshold function $t : V \to N$ where $1 \leq t(v) \leq d(v)$ for every $v \in V$
Output: A robust set $V' \subseteq V$ such that $|V'|$ is maximized.

If the threshold function is defined by $t(v) = d(v), \forall v \in V$ then we add the suffix With Unanimity to the problem name. The *majority threshold* is $t(v) = \lceil \frac{d(v)}{2} \rceil, \forall v \in V$.

Parameterized Complexity. Here we only give the basics notions on parameterized complexity, for more background the reader is referred to [11,18]. A decision problem parameterized by k is said to be *fixed-parameter tractable* if there exists an algorithm that solves every instance (I, k) in fpt-time *i.e.*, in $f(k).|I|^{O(1)}$-time for some function f depending solely on k.

The basic class of parameterized intractability is W[1] and there is a good reason to believe that W[1]-hard problems are unlikely to be fixed-parameter tractable. In fact, there is a all hierarchies of classes W[i] with the following inclusions FPT $\subseteq$ W[1] $\subseteq$ W[2] $\ldots$. Informally speaking, a problem in W[i] is considered "harder" than those in W[$i - 1$] where $i > 1$. These classes are defined via the satisfiability problem of boolean circuits. More specifically, a parameterized problem belongs to W[i] if every instance (I, k) can be transformed in fpt-time to a boolean circuit C of constant *depth* and *weft* at most i, such that (I, k) is a yes instance if and only if there is a satisfying truth assignment for C of weight exactly k. The *weft* of a circuit is the maximum number of large gates *i.e.*, gates with a number of inputs not bounded by any constant, on a path from an input to the output. The *depth* is the maximum number of all gates on a path from an input to the output.

In this paper, the kernel size is expressed in terms of the number of vertices.

Approximation. Given an optimization problem A and an instance I of this problem, we denote by $|I|$ the size of I, by $opt_A(I)$ the optimum value of I and by $val(I, S)$ the value of a feasible solution S of I. In this paper, we will make use of the following approximation preserving reduction.

Definition 1 (L-reduction [19]). *Let A and B be two optimization problems. Then A is said to be L-reducible to B if there are two constants $\alpha, \beta > 0$ and two polynomial time computable functions f, g such that*

1. *f maps an instance I of A into an instance I' of B such that $opt_B(I') \leq \alpha \cdot opt_A(I)$,*
2. *g maps solutions each solution S' of I' into a solution S of I such that $|val(I, S) - opt_A(I)| \leq \beta \cdot |val(I', S') - opt_B(I')|$.*

For us the important property of this reduction is that if A is APX-hard then B is also APX-hard.

3 Parameterized Complexity

In this section, we consider the parameterized complexity of k-Robust Set. In some reductions we make use of the following gadget: a *forbidden edge* denotes an edge uv where both vertices have threshold one. Attaching a forbidden edge to a vertex w means to create a forbidden edge uv and make w adjacent to u. Notice that none of the three vertices u, v or w can be part of a robust set.

First, we show that k-Robust Set belongs to W[2] using the *Turing way*, that is, we reduce k-Robust Set to the Short Multi-tape Nondeterministic Turing Machine problem that is proved to belong to W[2] in [6] and defined as follows: given a multi-tape nondeterministic Turing machine M, a word x on the input alphabet of M, and an integer k, determine if there is a computation of M on input x that reaches a final accepting state in at most k steps. The parameter is k.

Theorem 1. k-Robust Set *is in W[2].*

Proof. We construct an fpt-reduction from k-Robust Set to Short Multi-tape Nondeterministic Turing Machine as follows. Let (G, t, k) be an instance of k-Robust Set with $G = (V, E)$ and $V = \{v_1, \dots, v_n\}$. We construct the following Turing machine M from (G, t, k). We create $n+1$ tapes denoted by $T_0, T_{v_1} \dots, T_{v_n}$. The tapes alphabet is $V \cup \{\times, 1, \dots, n\}$ plus the blank symbol 0. Initially, every tape is filled with 0. The transition function is defined hereafter. The machine M starts by writing symbol $\times$ on tape T_0 and move T_0's head one step to the right. During the first phase, M non-deterministically chooses k vertices and write them on tape T_0, that is, if M picks up a vertex $v \in V$ then it writes symbol v on T_0 and move T_0's head one step to the right. The previous procedure is done in $k+1$ steps. During the second phase, M verifies that the selected set is a robust set as follows. First, the machine move T_0's head one step to the left. Assume that T_0's head reads symbol v and, for every $u \in N(v)$, T_u's head reads symbol s_u. If $s_u = t(u) - 1$ then M goes in rejecting state. Otherwise, M writes symbol $s_u + 1$ on tape T_u and moves the T_0's head one step to the left. We repeat the previous procedure until T_0's head reads symbol ×. Clearly, this checking phase is performed in at most $k + 1$ steps. Finally, the input word x is empty and $k' = 2k + 2$. It is not hard to see that (G, t, k) is a Yes-instance if and only if M accepts in at most k' steps. □

Now in order to prove the W[2]-hardness of k-Robust Set, we construct a simple fpt-reduction from the problem Red/Blue Dominating Set proved W[2]-hard in [11] and defined as follows: given a bipartite graph $G = (R \cup B, E)$ and a positive integer k, determine if there exists a set $R' \subseteq R$ of cardinality k such that every vertex in B has at least one neighbor in R'. The parameter is k.

Theorem 2. k-Robust Set *is W[2]-complete even for bipartite graphs.*

Proof. Membership follows from Theorem 1. Now, let us show the W[2]-hardness. Given (G, k) an instance of Red/Blue Dominating Set, we construct an instance $(G' = (V', E'), t, k)$ of k-Robust Set as follows. We consider the complement $\bar{G}$ of the graph G, that is two vertices $u \in R$ and $v \in B$ are adjacent in $\bar{G}$ if and only if they are not adjacent in G. Moreover, the sets R and B remain independent sets. Graph G' is obtained from this last graph by attaching $\max\{k - d_{\bar{G}}(v), 1\}$ forbidden edges to each vertex $v \in B$. Finally, set $t(v) = k$ for every vertex $v \in B$ and $t(v) = 1$ for every vertex $v \in R$. Adding several forbidden edges to the vertices of B make sure that the threshold of these vertices is less than or equal to their degree as required.

Assume that (G, k) has a solution $R' \subseteq R$ of size k. One can see that R' is also a solution for (G', t, k) since every vertex in B is not adjacent to at least one vertex in R'. Conversely, suppose that there is a robust set $S \subseteq V'$ of size k in G'. Since S is robust, S cannot contain any vertex from B because of the forbidden edges, and thus S is entirely contained in R. Moreover, every vertex v in B is adjacent in G' to at most $t(v) - 1 = k - 1$ vertices in S. Hence, every vertex in B is adjacent in G to at least one vertex in S. Therefore, S is a solution of size k for (G, k). □

In the next two theorems, we show that k-Robust Set goes one level down in the W-hierarchy when all thresholds are bounded by a constant.

Theorem 3. k-Robust Set *is in W[1] if all thresholds are constant bounded.*

Proof. Let (G, t, k) be an instance of k-Robust Set where $t(v) \leq c$, $\forall v \in V$ for some constant $c > 0$. We construct in $O(n^c)$-time, where n is the number of vertices of G, a boolean circuit C of depth 3 and weft 1 as follows. We identify the inputs of the circuit with the vertices of G. Connect a $\neg$-gate to every input. For all $v \in V$ and all subsets $S' \subseteq N(v)$ of size $t(v)$, add a $\vee$-gate connected to $\neg$-gate of inputs in S'. Finally, add a large $\wedge$-gate connected to every $\vee$-gate. It is not hard to see that G admits a robust set of size k if and only if there is a weight-k assignment that satisfies C. □

We establish the W[1]-hardness of k-Robust Set by an fpt-reduction from the problem Red/Blue NonBlocker [10] defined as follows: given a bipartite graph $G = (R \cup B, E)$ and a positive integer k, determine if there is a set $R' \subseteq R$ of cardinality k such that every vertex in B has at least one neighbor that does not belong to R'. The parameter is k. This problem remains W[1]-hard even when every vertex in B has degree two and every vertex of R has degree at least two [10].

Theorem 4. k-Robust Set *is W[1]-complete even*

1. *For bipartite graphs and constant majority threshold.*
2. *For split graphs and constant threshold $t(v) = 2, \forall v \in V$.*

Proof. Membership follows from Theorem 3. We now prove the W[1]-hardness.

(1): Let (G, k) be an instance of Red/Blue NonBlocker where vertices in B have degree two, we construct the graph $G' = (V', E')$ from G as follows. For each vertex $v \in B$, attach a forbidden edge to v. Set $t(v) = \lceil d_{G'}(v)/2 \rceil$ for all $v \in V'$.

Assume that (G, k) has a solution $R' \subseteq R$ of size k. It is not hard to see that R' is also a solution for (G', t, k). Conversely, suppose that there is a robust set $S \subseteq V'$ of size k in G'. Because of the forbidden edges, the set S is entirely contained in R. Since R is a robust set, every vertex in B is adjacent to at least one vertex in $R \setminus S$. Therefore, S is a solution of size k for (G, k).

(2): Let (G, k) be an instance of Red/Blue NonBlocker where vertices in B have degree two and every vertex of R has degree at least 2, we construct the graph $G' = (V', E')$ from G as follows. Add edges to make B a clique. Set $t(v) = 2$ for all $v \in V$ and $k' = k$. Without loss of generality we may assume that $k \geq 2$.

Assume that (G, k) has a solution $R' \subseteq R$ of size k. One can easily verify that R' is a robust set of size k' for (G', k'). Conversely, suppose that there is a robust set $S \subseteq V'$ of size k' in G'. Notice that $S \cap B = \emptyset$ since otherwise we would not have been able to take more than one vertex in G. Indeed, if there are two vertices $u, v \in S$ with $v \in B$ then there is always a vertex $w \in B - \{u, v\}$ adjacent to both v and u. Thus, S is entirely contained in R. From now, it is not hard to see that R is also a solution for (G, k). □

It is interesting to note that the ratio between the number of unbounded vertices and the number of bounded vertices of the graph in the proof of Theorem 2 can be made arbitrarily small (add many forbidden edges). This implies a sharp dichotomy between the W[2]- and W[1]-completeness of k-Robust Set regarding the thresholds.

Unanimity Threshold. We consider now the k-Robust Set With Unanimity problem. First, we start with the following easy observation. In the case of unanimous threshold, any robust set is the complement of a total dominating set. Recall that a total dominating set S is a set of vertices such that every vertex has at least one neighbor in S. Moreover, we have the following theorem.

Theorem 5. *[9] If G is a connected graph of order at least 3 then there is a total dominating set of size at most $2n/3$.*

This implies that MAX ROBUST SET WITH UNANIMITY always has a solution of size at least $n/3$ when $n \geq 3$. The consequence of this result is that we directly get a linear kernel of size $3k$. Indeed, let (G, t, k) be an instance of k-ROBUST SET WITH UNANIMITY, if $k \leq n/3$ then the answer is YES. If $k > n/3$ then the instance (G, t, k) is a kernel of size at most $3k$. However, the parameter k is "large" in this last case. This suggests to look for other parameterizations.

Parametric Dual. Now, we show that $(n-k)$-ROBUST SET is FPT with respect to the parameter k for a large family of threshold functions.

Reduction rule 1. *Let (G, t, k) be an instance of $(n-k)$-*ROBUST SET*. If there is a vertex v such that $d(v) \geq k + t(v) - 1$ then remove v and decrease by one the threshold of every vertex in $N(v)$ to get a new equivalent instance (G', t', k).*

Lemma 1. *Reduction rule 1 is sound.*

Proof. Let $S \subseteq V$ be a robust set of size at least $n-k$. If there is a vertex v with $d(v) \geq k + t(v) - 1$ then v must be in S since otherwise v has at most $k-1$ neighbors in $V \setminus S$ and then at least $t(v)$ neighbors in S. □

Theorem 6. $(n-k)$-ROBUST SET *admits a kernel of size $O(k^2)$ if for all $v \in V$ $t(v) = \lceil \alpha_v d(v)^{\beta_v} + \gamma_v \rceil$ for any constants $\alpha_v, \beta_v \in [0, 1]$, $\alpha_v \beta_v \neq 1$, and $\gamma_v \in Q$.*

Proof. Let (G, t, k) be an instance of $(n-k)$-ROBUST SET. Exhaustively apply reduction rule 1 to get (G', t', k). Assume that there exists a solution $S \subseteq V$ of size at least $n-k$. Because of reduction rule 1, we have that

$$d(v) < k + t(v) - 1 = k + \lceil \alpha_v d(v)^{\beta_v} + \gamma_v \rceil - 1 \leq k + \alpha_v d(v)^{\beta_v} + \gamma_v$$

We claim that $d(v) \leq \theta_v(k)$ for all $v \in V'$ where $\theta_v(k) = \frac{k+\gamma_v}{1-\alpha_v} + (1/\beta_v)^{\frac{1}{1-\beta_v}}$ if $\alpha_v \neq 1$, $\frac{k+\gamma_v}{1-\beta_v} + (1/\beta_v)^{\frac{1}{1-\beta_v}}$ otherwise. Consider the following cases.

Case 1. If $\beta_v = 0$ then obviously $d(v) \leq \theta_v(k)$

Case 2. If $\beta_v = 1$ then $d(v) < \frac{k+\gamma_v}{1-\alpha_v} < \theta_v(k)$ (since $\alpha_v < 1$)

Case 3. Suppose now that $\beta_v \in (0, 1)$. First, it is not hard to show that the following holds: $n^{\beta_v} \leq \beta_v n$ if and only if $n \geq (1/\beta_v)^{\frac{1}{1-\beta_v}}$ for any $n \geq 1$ and $\beta_v \in (0, 1)$. Hence, If $d(v) \geq (1/\beta_v)^{\frac{1}{1-\beta_v}}$ then we have $d(v) \leq k + \alpha_v \beta_v d(v) + \gamma_v$ and thus $d(v) \leq \frac{k+\gamma_v}{1-\alpha_v\beta_v} \leq \theta_v(k)$. Otherwise $d(v) < (1/\beta_v)^{\frac{1}{1-\beta_v}} \leq \theta_v(k)$.

Since every vertex from S has at least one neighbor in $V' - S$ then $|S|$ has at most $|V' - S| d^{max} \leq k\theta^{max}(k)$ vertices where $\theta^{max}(k) = \max_{v \in V'} \theta_v(k)$ and d^{max} is the maximum degree of vertices in $V' - S$.

The kernelization procedure is then defined as follows. From an instance (G, t, k) of $(n-k)$-ROBUST SET, exhaustively apply reduction rule 1 to get an instance (G', t', k). If $|V'| > k\theta^{max}(k) + k$ then return a trivial NO-instance. Otherwise, return the instance (G', t', k). □

Notice that if $\alpha_v = \beta_v = 1$ and $\gamma_v = 0$, $\forall v \in V$ then the $(n-k)$-ROBUST SET problem is exactly the TOTAL DOMINATING SET problem which is known to be W[2]-hard [14].

4 Algorithm for Tree-Like Graphs

In this section we establish a $O(T^{2\omega}n)$-time algorithm for Max Robust Set and a $O((k+1)^{2\omega}n)$-time algorithm for k-Robust Set where T is the maximum threshold and ω the treewidth of the input graph.

Using a nice tree decomposition together with a dynamic programming algorithm we can prove the following.

Theorem 7. Max Robust Set *is solvable in time* $O(T^{2\omega}n)$ *where* T *is the maximum threshold and* ω *is the treewidth of the input graph.*

Now we show that k-Robust Set is solvable in $O((k+1)^{2\omega}n)$ time. For that purpose, we introduce the following reduction rule.

Reduction rule 2. *Let* (G, t, k) *be an instance of* k-Robust Set*. If there is a vertex* v *such that* $t(v) > k+1$ *then set the threshold* $t(v)$ *to* $k+1$ *to get a new equivalent instance* (G, t', k)*.*

Lemma 2. *Reduction rule 2 is sound.*

Proof. Let $(G = (V, E), k, t)$ be an instance of k-Robust Set. Exhaustively apply Reduction Rule 2 on (G, t, k) to get a new instance (G, t, k'). It is not hard to see that if $S \subseteq V$ is a robust set of size at least k for (G, t, k), then any subset of size k of S is a robust set for (G, t, k'). The converse is clear. □

We are now ready to prove the following.

Theorem 8. k-Robust Set *is solvable in time* $O((k+1)^{2\omega}n)$ *where* ω *is the treewidth of the input graph.*

Proof. Let (G, t, k) be an instance of k-Robust Set. Exhaustively apply Reduction Rule 2 on (G, t, k) to get a new instance (G, t, k'). Apply the algorithm from Theorem 7 on (G, t') to get the optimal solution of value *opt*. If *opt* $\geq k$ return Yes; otherwise return No. Since every threshold is at most $k+1$, the running time is $O((k+1)^{2\omega}n)$. □

Notice that if all thresholds are constant bounded then k-Robust Set is in FPT with respect to the parameter treewidth.

5 Approximability

In this section, we show that Max Robust Set is inapproximable even for small constant thresholds. In order to prove this result, we consider the Max Clique problem: given a graph $G = (V, E)$, find a clique $C \subseteq V$ of maximum size.

Theorem 9. *If* $NP \neq ZPP$, Max Robust Set *is not approximable within* n^{ϵ} *for any* $\epsilon > 0$ *even for thresholds at most two.*

We now prove the APX-completeness of Max Robust Set With Unanimity.

Lemma 3. Max Robust Set With Unanimity *is* 3*-approximable in polynomial time.*

Proof. The algorithm consists of the following two steps:

1. Compute a spanning tree T of G.
2. Compute an optimal solution S of T.

Using Theorem 7, the algorithm runs in polynomial-time. Clearly, any feasible solution for T is also a solution for G. Moreover, using Theorem 5, we have $|S| \geq n/3 \geq opt(G)/3$. □

Theorem 10. Max Robust Set With Unanimity *is APX-complete.*

Proof. Membership follows from Lemma 3. In order to prove the APX-hardness we provide an L-reduction (see Definition 1) from Max E2Sat-3 proved APX-hard in [4] and defined as follows: given a CNF formula ϕ with n variables and m clauses, in which every clause contains exactly two literals and every variable appears in exactly three clauses, determine an assignment to the variables satisfying a maximum number of clauses. Notice that $m = 3n/2$.

Given a formula ϕ of Max E2Sat-3, we construct an instance $I = (G = (V, E), t, k)$ of Max Robust Set With Unanimity as follows (see Figure 1). For every variable x_i, we construct the complete bipartite graph $K_{3,3}(x_i) = (V^-(x_i), V^+(x_i))$ in which every edge uv is replaced by an edge-vertex e_{uv} and two edges ue_{uv} and $e_{uv}v$. We denote by $E(x_i)$ this set of edge-vertices. The vertices in $V^+(x_i)$ (resp. $V^-(x_i)$) represents the positive (resp. negative) literals of x_i. We denote by A the set of all vertices added so far. For every clause c_j in ϕ add two adjacent clause-vertices $\bar{c}_j$ and $\bar{c}'_j$. For every variable x_i, if x_i appears positively (resp. negatively) in a clause c_j then add an edge between $\bar{c}'_j$ and a vertex of $V^-(x_i)$ (resp. $V^+(x_i)$). Thus, vertex $\bar{c}_j$ represents the complement of the clause c_j in ϕ. Finally, add two adjacent vertices c and c'. For every vertex $v \in V^-(x_i) \cup V^+(x_i)$, if v is not adjacent to a clause-vertex then add the edge vc'.

The optimal value in I is bounded by the number of vertices of G and thus, $opt(I) \leq 15n + 2m + 2 \leq 16opt(\phi) + 2 \leq 18opt(\phi)$ since $opt(\phi) \geq 3/4m$ and $opt(\phi) \geq 1$.

Moreover, let $x^* \subseteq V$ be an optimal assignment for ϕ and let

$$S = \cup_{x_i^*=1} V^+(x_i) \cup \cup_{x_i^*=0} V^-(x_i) \cup \cup_{i=1}^{n} E(x_i) \cup \{\bar{c}_j : c_j \text{ is satisfied by } x^*\} \cup \{c\}.$$

We can easily verify that S is a robust set and $|S \cap (V^-(x_i) \cup V^+(x_i) \cup E(x_i)| = 12$ and thus $|S \cap A| = 8m$ and then $opt(I) \geq |S| = 8m + opt(\phi) + 1$.

Let S be a robust set for I. We show in the following how to construct an assignment a_S for ϕ from the solution S such that $val(\phi, a_S) = |S| - 8m - 1$. For each variable x_i, S cannot contain vertices from both $V^-(x_i)$ and $V^+(x_i)$ since otherwise an edge-vertex has both neighbors inside S. Notice also that S

cannot contain any vertex $\bar{c}'_j$, since $\bar{c}'_j$ is adjacent to the degree one vertex $\bar{c}_j$. Similarly $c' \notin S$.

If S contains for every $i = 1, \ldots, n$ the set $E(x_i)$ and one of the sets $V^-(x_i)$ or $V^+(x_i)$ then $|S \cap A| = 8m$ and we can defined the following assignment a_S: $x_i = 1 \Leftrightarrow |S \cap V^+(x_i)| \neq 0$. In this case, a clause-vertex is in S if and only if the corresponding clause is satisfied by a_S. Thus, the number of clauses satisfied by a_S is exactly $val(\phi, a_S) = |S| - 8m - 1$.

Assume now that $|S \cap A| < 8m$. We show that there exists an other solution S' with $|S'| \geq |S|$ such that $|S' \cap A| = 8m$. If a vertex $v \in E(x_i) \setminus S$ for some $i \in \{1, \ldots, n\}$, we can add v in S since v cannot have both neighbors in S. Similarly, if c is not in S, then we add c in S.

Since $|S \cap A| < 8m$, there is at least one vertex either in $V^+(x_i) \setminus S$ or in $V^-(x_i) \setminus S$ for some $i \in \{1, \ldots, n\}$. Without loss of generality, we only consider vertices in $V^+(x_i) \setminus S$. A vertex $v \in V^+(x_i) \setminus S$ is either adjacent to c' or to a clause-vertex $\bar{c}'_j$. In the first case we add v in S. In the second case, we denote $N(\bar{c}'_j) = \{v, v', \bar{c}_j\}$. If $v' \in S$ and $\bar{c}_j \in S$ then remove $\bar{c}_j$ from S and add v instead, otherwise, add v in S. Thus, we obtain a new solution S' such that $|S'| \geq |S|$ and $|S' \cap A| = 8m$ and in this case as below, we can obtain an assignment $a_{S'}$ such that $|S'| - val(\phi, a_{S'}) = 8m + 1$. In particular, if S' is an optimal solution, then $opt(\phi) \geq val(\phi, a_{S'}) = opt(I) - 8m - 1$ and thus, we have $opt(I) - opt(\phi) = 8m + 1$ and then $opt(\phi) - val(\phi, a_{S'}) = opt(I) - |S'|$. □

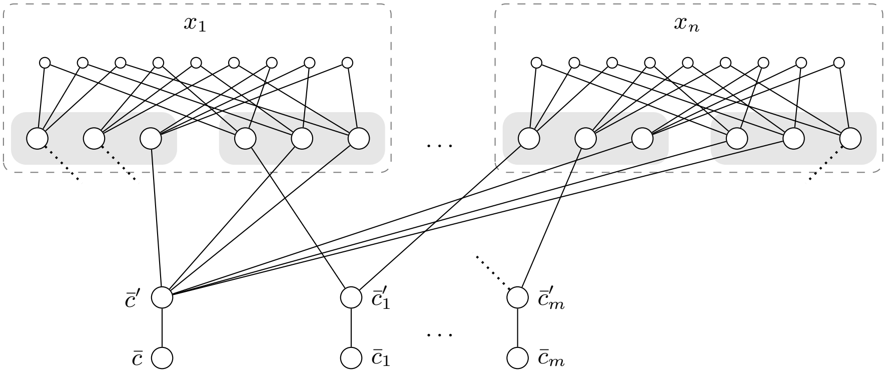


Fig. 1. The construction of G

In the following we propose a PTAS on the class of planar graphs, using the polynomial time algorithm for graphs of bounded treewidth.

Theorem 11. Max Robust Set *on planar graphs admits a PTAS.*

Proof. Given a planar embedding of an input graph, we consider the set of the vertices which are on the exterior face, they will be called *level 1 vertices.* By induction we define *level* k as the vertices which are on the exterior face when we have removed the vertices of levels smaller than k [2]. A planar embedding is k-*level* if it has no nodes of level greater than k. If a planar graph is k-level, it has a k-outerplanar embedding.

If we want to achieve an approximation within $1+\varepsilon$, let us consider $k = 2(1+\lceil\frac{1}{\varepsilon}\rceil)$. Let X_t be the set of vertices of level t and let H_i, $0 \leq i \leq k-1$, be the graph obtained from G by considering the subgraphs formed by the set of vertices $\bigcup_{t+1\leq j\leq t+k} X_j$, for $t \equiv i \pmod{(k-2)}$. The subgraph containing exactly $\bigcup_{t+1\leq j\leq t+k} X_j$ is k-outerplanar, and so is H_i, too.

Since H_i is k-outerplanar, it has treewidth at most $3k-1$ [5]. We construct graph H_i' from H_i by attaching a forbidden edge to each vertex on the boundary (that means vertices in $X_{t+1}, X_{t+2}, X_{t+k-1}X_{t+k}$ with $t \equiv i (\text{mod } (k-2))$). Thus, in each subgraph of H_i' the vertices in $X_{t+1}, X_{t+2}, X_{t+k-1}X_{t+k}$ cannot take part from any robust set.

On applying Theorem 7, we can efficiently determine an optimal robust set in each subgraph of H_i'. Denote by S_i the union of these robust sets. Clearly S_i is a robust set on H_i.

Among $S_0, \ldots, S_{k-1}$ we choose the best solution that we denote S and we are going to prove that S is an $(1+\varepsilon)$-approximation of the optimal value on G. We can easily show that there is at least one r, $0 \leq r \leq k-1$ such that at most $\frac{2}{k}$ of vertices in an optimal solution S_{opt} of G are on levels $X_{t+1}, X_{t+2}, X_{t+k-1}X_{t+k}$ with $t \equiv r\, (\text{mod } (k-2))$). This means that the solution S_r obtained by deleting the vertices from levels $X_{t+1}, X_{t+2}, X_{t+k-1}X_{t+k}$ from S_{opt} will have at least $|S_{opt}|(1-\frac{2}{k}) = \frac{k-2}{k} opt$ vertices. According to our algorithm, $|S| \geq |S_r| \geq \frac{opt}{1+\varepsilon}$.

The overall running time of the algorithm is k times what we need for graphs of treewidth at most k, that is $O(kT^{6k-2}n) = n^{O(1/\varepsilon)}$ where $T = \max_{v\in V}$ $t(v)$. □

6 Conclusion

In this paper, we introduced the k-Robust Set problem. We established positive and negative results concerning its parameterized tractability and approximability. However, several questions remain open. For instance, we do not know if the problem is fixed-parameter tractable for parameter treewidth. Another interesting open question is whether k-Robust Set With Unanimity is fixed-parameter tractable for parameter k when we ask to determine the existence of a robust set of size at least $\lceil\frac{n}{3}\rceil + k$. Finally, there is room enough for improving the approximability of Max Robust Set With Unanimity.

References

1. Aazami, A., Stilp, M.D.: Approximation Algorithms and Hardness for Domination with Propagation. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 1–15. Springer, Heidelberg (2007)

2. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. Journal of the ACM 41(1), 153–180 (1994)
3. Ben-Zwi, O., Hermelin, D., Lokshtanov, D., Newman, I.: Treewidth governs the complexity of Target Set Selection. Discrete Optimization 8(1), 87–96 (2011)
4. Berman, P., Karpinski, M.: On Some Tighter Inapproximability Results. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 200–209. Springer, Heidelberg (1999)
5. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science 209(1-2), 1–45 (1998)
6. Cesati, M.: The Turing way to parameterized complexity. Journal of Computer and System Sciences 67(4), 654–685 (2003)
7. Chalermsook, P., Chuzhoy, J.: Resource minimization for fire containment. In: Proceedings of the 21th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 1334–1349 (2010)
8. Chen, N.: On the approximability of influence in social networks. In: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008), pp. 1029–1037 (2008)
9. Cockayne, E.J., Dawes, R.M., Hedetniemi, S.T.: Total domination in graphs. Networks 10(3), 211–219 (1980)
10. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness II: On completeness for W[1]. Theoretical Computer Science 141(1-2), 109–131 (1995)
11. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer, New York (1999)
12. Dreyer Jr., P.A., Roberts, F.S.: Irreversible k-threshold processes: Graph-theoretical threshold models of the spread of disease and of opinion. Discrete Applied Mathematics 157(7), 1615–1627 (2009)
13. Finbow, S., MacGillivray, G.: The firefighter problem: a survey of results, directions and questions. The Australasian Journal of Combinatorics 43, 57–77 (2009)
14. Golovach, P.A., Kratochvil, J., Suchý, O.: Parameterized complexity of generalized domination problems. Discrete Applied Mathematics 160(6), 780–792 (2012)
15. Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003), pp. 137–146 (2003)
16. Kimura, M., Saito, K., Motoda, H.: Blocking links to minimize contamination spread in a social network. ACM Transactions on Knowledge Discovery from Data 3(2), 9:1–9:23 (2009)
17. Nichterlein, A., Niedermeier, R., Uhlmann, J., Weller, M.: On Tractable Cases of Target Set Selection. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010. LNCS, vol. 6506, pp. 378–389. Springer, Heidelberg (2010)
18. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
19. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. J. Comput. Syst. Sci. 43(3), 425–440 (1991)

Mortality for 2×2 Matrices Is NP-Hard

Paul C. Bell[1], Mika Hirvensalo[2], and Igor Potapov[3]

[1] Department of Computer Science, Loughborough University, Loughborough,
LE11 3TU, UK
p.bell@lboro.ac.uk

[2] Department of Mathematics, University of Turku, FIN-20014 Turku, Finland
mikhirve@utu.fi

[3] Department of Computer Science, University of Liverpool, Ashton Building,
Ashton St, Liverpool, L69 3BX, UK
potapov@liverpool.ac.uk

Abstract. We study the computational complexity of determining whether the zero matrix belongs to a finitely generated semigroup of two dimensional integer matrices (the mortality problem). We show that this problem is NP-hard to decide in the two-dimensional case by using a new encoding and properties of the projective special linear group. The decidability of the mortality problem in two dimensions remains a long standing open problem although in dimension three is known to be undecidable as was shown by Paterson in 1970.

We also show a lower bound on the minimum length solution to the Mortality Problem, which is exponential in the number of matrices of the generator set and the maximal element of the matrices.

1 Introduction

In this paper we study the computational complexity of the problem of determining whether the zero matrix belongs to a matrix semigroup (the Mortality Problem) generated by a finite set of 2×2 integral matrices. The Mortality Problem of 3×3 integer matrices was shown to be undecidable in 1970 [13] and the question about 2×2 matrices is currently open.

In the past there has been much interest in decidability questions for problems concerning matrix semigroups and in particular the Mortality Problem [10,11], which have a number of connections with linear algebra, geometry and controllability of switched linear systems [7,6]. The mortality problem was shown to be decidable for a pair of rational 2×2 matrices in [7]. Also, it was recently shown in [12] that the Mortality Problem is decidable for any set of 2×2 integer matrices whose determinants assume the values $0, \pm 1$, by adapting a technique from [9]. The main goal of this paper is to show that the Mortality Problem for the same set of 2×2 integer matrices (whose determinants assume the values $0, \pm 1$) is NP-hard.

Another set of hardness results is known about bounded membership. A set of matrices over the integers is said to be k-mortal (with k a positive integer) if the zero matrix can be expressed as a product of length k of matrices in the set. In

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 148–159, 2012.

[5], it was shown that the bounded membership problem for the zero matrix (the k-mortality problem) is NP-hard for semigroups generated by a pair of matrices (where the dimension is variable). Also a straightforward encoding of the Subset Sum Problem can be used to show NP-hardeness of the bounded membership problem for 2×2 matrices including the case of commutative matrices [8].

In this paper we prove that the *unbounded* mortality problem for 2×2 matrices is NP-hard by a more sophisticated construction that requires detailed analysis of the semigroup generator as well as the use of an extended group alphabet and the concept of border letters. We also show a lower bound on the minimum length solution to the Mortality Problem, which is exponential in the number of matrices of the generator set and the maximal element of the matrices.

It is known that many computational problems for matrix semigroups and groups are inherently difficult to solve even for low-dimensions. In contrast to the Mortality(3×3), which was one of the first matrix problems, shown to be undecidable several decades ago, the *Identity Problem* [1] was shown to be undecidable for dimension 4 only a few years ago [2]. Moreover it has been recently proven in [9] that the Identity Problem for integral matrices of dimension 2 is decidable and later in [3] that the problem for $\mathrm{SL}_2(\mathbb{Z})$ is NP-hard. The NP-hardness result of this paper about the Mortality(2×2) corresponds very well with the Identity situation. Unfortunately, the same proof technique as in [3] cannot be directly applied for the Mortality Problem and therefore we must use new encoding and properties of the projective special linear group in this paper to derive the result.

2 Notations and the Structure of $\mathrm{SL}_2(\mathbb{Z})$

By an alphabet we understand (usually) a finite set Γ, and call its elements letters. Any alphabet can be furnished with algebraic structure, defining the product by letter juxtaposition (concatenation). Assumption that there are no nontrivial relations between the letters is another way to say that the alphabet generates a free monoid, denoted as Γ^* or $\langle \Gamma \rangle$. An element of the monoid Γ^* is called word, and the identity element is called *empty word* and denoted by ε or 1. A *group alphabet* is an alphabet augmented with inverse elements: $\Sigma = \{z_1, z_2, \ldots, z_k, \overline{z_1}, \overline{z_2}, \ldots, \overline{z_k}\}$, where z_i and $\overline{z_i}$ (notation $\overline{z_i} = z_i^{-1}$ is also used) are assumed to satisfy $z_i\overline{z_i} = \overline{z_i}z_i = \varepsilon$. The relation between a letter and its inverse is the only nontrivial relation in a group alphabet. We denote $\Sigma^+ = \{a_1 \ldots a_n \mid a_i \in \Sigma, n \geq 1\}$. For a word $w = w_1 w_2 \cdots w_n$, we denote $\overline{w} = w^{-1} = \overline{w_n} \cdots \overline{w_2}\,\overline{w_1}$.

Let Σ be a group alphabet. Using the notation of [1], we shall also introduce a reduction mapping which removes factors of the form $z\overline{z}$ for $z \in \Sigma$. To that end, we define the relation $\vdash \subseteq \Sigma^* \times \Sigma^*$ such that for all $w, w' \in \Sigma^*$, $w \vdash w'$ if and only if there exists $u, v \in \Sigma^*$ and $z \in \Sigma$ where $w = uz\overline{z}v$ and $w' = uv$. We

[1] The Identity Problem for matrix semigroups is a well-known challenging problem which is also equivalent to another fundamental problem in Group Theory: given a finitely generated matrix semigroup S, decide whether a subset of the generator of S generates a nontrivial group (Group Problem).

may then define by $\vdash^*$ the reflexive and transitive closure of $\vdash$. The following Lemma is well-known, see eg. [1] for the proof.

Lemma 1. *For each $w \in \Sigma^*$ there exists exactly one word $r(w) \in \Sigma^*$ such that $w \vdash^* r(w)$ does not contain any factor of the form $z\overline{z}$, with $z \in \Sigma$.*

The word $r(w)$ is called the reduced representation of word $w \in \Sigma^*$. As an example, we see that if $w = 132\overline{2}1\overline{1}\,\overline{3}\,\overline{1} \in \Sigma^*$, then $r(w) = \varepsilon$.

A homomorphism $h : \Gamma_1^* \to \Gamma_2^*$, between two monoids Γ_1^* and Γ_2^* is a mapping satisfying $h(ab) = h(a)h(b)$ for any $a, b \in \Gamma^*$ and $h(1) = 1$ where 1 denotes the identity element of the respective monoid. An injective homomorphism, h', is called a *monomorphism* and is denoted $\Gamma_1^* \hookrightarrow \Gamma_2^*$.

Notation $\mathbb{Z}^{2\times 2}$ stands for the set of all 2×2 integer matrices. This set has a natural ring structure with respect to ordinary matrix addition and multiplication. A subset of $\mathbb{Z}^{2\times 2}$, $\mathrm{GL}_2(\mathbb{Z})$ (also denoted as $\mathrm{GL}(2,\mathbb{Z})$) stands for the *general linear group* over the ring of integers, meaning all 2×2 integer matrices having integer matrix inverses:

$$\mathrm{GL}_2(\mathbb{Z}) = \{A \in \mathbb{Z}^{2\times 2} \mid \det(A) \in \{-1, 1\}\}.$$

Group $\mathrm{GL}_2(\mathbb{Z})$ is clearly the largest multiplicative matrix group contained in $\mathbb{Z}^{2\times 2}$, but quite often it is useful to study its subgroup $\mathrm{SL}_2(\mathbb{Z})$ (also denoted as $\mathrm{SL}(2,\mathbb{Z})$), the *special linear group* defined as

$$\mathrm{SL}_2(\mathbb{Z}) = \{A \in \mathrm{GL}_2(\mathbb{Z}) \mid \det(A) = 1\}.$$

Furthermore, it turns out that the quotient group

$$\mathrm{PSL}_2(\mathbb{Z}) = \mathrm{SL}_2(\mathbb{Z})/\{\pm I\},$$

called the *projective special linear group* has a very useful representation as a free product of two cyclic groups of order 2 and 3.

Group $\mathrm{SL}_2(\mathbb{Z})$ is very important in number theory, and its structure has been studied extensively in various textbooks (see [14], for instance), but for pointing out the algorithmic complexity issues, we reproduce the structural properties most relevant to our study here.

Two structurally important elements of $\mathrm{SL}_2(\mathbb{Z})$ are

$$S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Evidently $S^2 = -I$ (which implies $S^3 = -S$ and $S^4 = I$, so S has order 4), whereas for each $n \in \mathbb{Z}$,

$$T^n = \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix},$$

implying that T has no finite order.

Lemma 2. $\mathrm{SL}_2(\mathbb{Z}) = \langle S, T\rangle$. *Furthermore, any matrix*

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{SL}_2(\mathbb{Z})$$

can be represented as

$$A = S^{\gamma} T^{q_1} S^3 T^{q_2} S^3 \cdot \ldots S^3 T^{q_k} S^3 T^{q_{k+1}}, \tag{1}$$

so that $\gamma \in \{0, 1, 2, 3\}$, $q_i \in \mathbb{Z}$ *for some* $k \geq 0$.

It is worth noticing that even though all matrices $A \in \mathrm{SL}_2(\mathbb{Z})$ can be represented in terms of S and T, the representation is by no means unique. A direct computation shows that, for example,

$$TST = ST^{-1}S^3.$$

For a more canonical representation, let

$$R = ST = \begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix}.$$

Direct computation shows that

$$R^2 = \begin{pmatrix} -1 & -1 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad R^3 = -I,$$

implying that $R^6 = I$, so R is of order 6. Since now $T = S^{-1}R = S^3R$, it follows that $\mathrm{SL}_2(\mathbb{Z}) = \langle S, R\rangle$, and that a representation of $A \in \mathrm{SL}_2(\mathbb{Z})$ in terms of R and S can be obtained by substituting $T = S^3R = -SR$ in (1). It is noteworthy that when substituting $T = -SR$ in (1), one can use $R^3 = -I$ and $S^2 = -I$ to get a representation

$$A = (-1)^{\gamma'} R^{n_0} S R^{n_1} S \cdot \ldots \cdot R^{n_{l-1}} S R^{n_l}, \tag{2}$$

where $\gamma' \in \{0, 1\}$, $n_i \in \{0, 1, 2\}$ and $n_i \in \{1, 2\}$ for $0 < i < l$. It turns out, that representation (2) for a given matrix $A \in \mathrm{SL}_2(\mathbb{Z})$ is unique, but it is very common to present this result ignoring the sign. For that purpose, we let $s = S\{\pm I\}$ and $r = R\{\pm I\}$ be the projections of S and R in $\mathrm{PSL}_2(\mathbb{Z})$.

Lemma 3. $\mathrm{PSL}_2(\mathbb{Z})$ *is a free product of* $\langle s\rangle = \{1, s\}$ *and* $\langle r\rangle = \{1, r, r^2\}$. *That is, if*

$$r^{n_0} s r^{n_1} s \cdot \ldots \cdot r^{n_{p-1}} s r^{n_p} = r^{m_0} s r^{m_1} s \cdot \ldots \cdot r^{m_{q-1}} s r^{m_q},$$

where $n_i, m_j \in \{0, 1, 2\}$ *and* n_i, $m_j \in \{1, 2\}$ *for* $0 < i < p$ *and* $0 < j < q$, *then* $p = q$ *and* $n_k = m_k$ *for each* $0 \leq k \leq p$.

For the proof of the lemma see [14]. We say that a representation in $\mathrm{PSL}_2(\mathbb{Z})$ is *reduced* if it satisfies the conditions of the previous lemma.

Mortality Problem: Decide whether a given finitely generated matrix semigroup contains the zero matrix.

3 The Mortality Problem

In this section we show that the mortality problem is NP-hard for a finite set of matrices from $\mathbb{Z}^{2\times 2}$. In order to prove this result, we shall adapt the proof technique used in [3] by showing a monomorphism (injective homomorphism) between an arbitrary sized alphabet and $\mathrm{PSL}_2(\mathbb{Z})$ with certain essential properties.

Lemma 4. *Given a group alphabet $\Sigma = \{z_1, z_2, \ldots, z_k, \overline{z_1}, \overline{z_2}, \ldots, \overline{z_k}\}$ and a binary group alphabet $\Sigma_2 = \{c, d, \overline{c}, \overline{d}\}$, then mapping $\alpha : \Sigma \to \Sigma_2^*$ defined by*

$$\alpha(z_i) = c^i d \overline{c}^i, \quad \alpha(\overline{z_i}) = c^i \overline{d} \overline{c}^i$$

can be extended to a monomorphism $\alpha : \Sigma^* \hookrightarrow \Sigma_2^*$, see [4] for more details.

Lemma 5. *Let $\Sigma_2 = \{c, d, \overline{c}, \overline{d}\}$ be a group alphabet. Then the mapping $\beta : \Sigma_2 \to \mathrm{PSL}_2(\mathbb{Z})$ defined by:*

$$\beta(c) = (rsr)^2, \quad \beta(d) = (rs)^2, \quad \beta(\overline{c}) = (r^2sr^2)^2, \quad \beta(\overline{d}) = (sr^2)^2$$

can be extended to a monomorphism $\beta : \Sigma_2^* \hookrightarrow \mathrm{PSL}_2(\mathbb{Z})$.

Proof. Recall that $\mathrm{PSL}_2(\mathbb{Z})$ has monoid presentation $\langle s, r | s^2 = r^3 = 1\rangle$, and let $w \in \{c, d, \overline{c}, \overline{d}\}^*$ be a reduced word (i.e., contains no subwords in $\{c\overline{c}, \overline{c}c, d\overline{d}, \overline{d}d\}$). It suffices to show that from $\beta(w)$, we can always deduce the initial symbol of $w \in \Sigma_2^*$.

This follows from a case analysis. The possible initial parts of w can be discovered by computing all products $\beta(cc) = (rsr)^4 = rsr^2sr^2sr^2sr$, $\beta(cd) = (rsr)^2(rs)^2 = rsr^2sr^2srs$, $\beta(c\overline{d}) = rsr^2srsr^2sr^2$, $\beta(dc) = rsrsrsr^2sr$, $\beta(dd) = rsrsrsrs$, $\beta(d\overline{c}) = rsrsr^2srsr^2$, $\beta(\overline{c}d) = r^2sr^2s$, $\beta(\overline{cc}) = r^2srsrsrsr^2$, $\beta(\overline{c}\overline{d}) = r^2srsr^2sr^2sr^2$, $\beta(\overline{d}c) = srsr$, $\beta(\overline{d}\overline{c}) = sr^2srsrsr^2$, and $\beta(\overline{dd}) = sr^2sr^2sr^2sr^2$. The claim follows now from comparing the initial parts of the sequences (no-one is a prefix of another) and from the fact that $\mathrm{PSL}_2(\mathbb{Z}) = \langle s\rangle * \langle r\rangle$. □

Example 1. Let $w \in \{c, d, \overline{c}, \overline{d}\}$ and

$$\beta(w) = r^2sr^2srsr^2sr^2sr^2srsrsr^2srsrsrs.$$

As representation in $\mathrm{PSL}_2(\mathbb{Z})$ in this form is unique, we must conclude that w begins with $\overline{c}$ (Only the image of $\overline{c}$ begins with r^2). To proceed, we introduce identity $1 = r^2sr^2rsr$ in the representation of $\beta(w)$ to see that

$$\begin{aligned}\beta(w) &= r^2sr^2(r^2sr^2rsr)srsr^2sr^2sr^2srsrsr^2srsrsrs\\ &= (r^2sr^2r^2sr^2)rsrsrsr^2sr^2sr^2srsrsr^2srsrsrs\\ &= \beta(\overline{c})rsrsrsr^2sr^2sr^2srsrsr^2srsrsrs.\end{aligned}$$

writing $w = \overline{c}w_1$ we see that

$$\beta(w_1) = rsrsrsr^2sr^2sr^2srsrsr^2srsrsrs,$$

and now we must have $w_1 = dw_2$, and

$$\beta(w_2) = rsr^2sr^2sr^2srsrsr^2srsrsrs$$

continuing in the same way we see that $w_2 = cw_3$,

$$\beta(w_3) = rsr^2srsrsr^2srsrsrs,$$

$w_3 = cw_4$,

$$\beta(w_4) = srsr^2srsrsrs,$$

and now w_4 must begin with $\overline{d}$. Again introducing identity $1 = rsr^2rsr^2$ we see that

$$\beta(w_4) = sr(rsr^2rsr^2)sr^2srsrsrs = (srrsr^2)rsr^2sr^2srsrsrs,$$

and if $w_4 = \overline{d}w_5$, then

$$\beta(w_5) = rsr^2sr^2srsrsrs,$$

and letting $w_5 = cw_6$

$$\beta(w_6) = rsrsrsrs = \beta(dd)$$

combining all letters we see that $w = \overline{c}dcc\overline{d}cdd$.

Lemma 6. *For any nonempty reduced word $w \in \Sigma^+$, $\beta \circ \alpha(w)$ has first letter r and last letter r in its reduced representation under* $\mathrm{PSL}_2(\mathbb{Z})$*, where $\beta \circ \alpha : \Sigma^* \to \mathrm{PSL}_2(\mathbb{Z})$.*

Proof. For each letter z_i we have

$$\beta(\alpha(z_i)) = \beta(c^i d\overline{c}^i) = \beta(c)^i\beta(d)\beta(\overline{c})^i.$$

Now that $\beta(c)$ begins with r, so does $\beta(\alpha(z_i))$, for otherwise $\beta(\alpha(z_i))$ would have two representations, $\beta(\alpha(z_i)) = r\ldots$ and $\beta(\alpha(z_i)) = s\ldots$, contradicting Lemma 5. Similar conclusion can be made for the ending letter and for $\beta(\alpha(\overline{z_i}))$, as well. The result can be directly extended to words $w \in \Sigma^+$, as the failure of it would contradict Lemma 5 as well. □

We see that $\beta\circ\alpha : \Sigma^* \hookrightarrow \mathrm{PSL}_2(\mathbb{Z})$ is a monomorphism since it is the composition of two monomorphisms. We require the following lemma concerning the size of the matrix when $\beta \circ \alpha$ is applied to the power of a letter from Σ.

Lemma 7. *Given $\Sigma = \{z_1, z_2, \ldots, z_k, \overline{z_1}, \overline{z_2}, \ldots, \overline{z_k}\}$, for any letter $z_i \in \Sigma$:*

$$\beta \circ \alpha(z_i^j) = \{\pm I\}\begin{pmatrix} -8i^2j - 4ij - 1 & -8i^2j \\ 8i^2j + 8ij + 2j & 8i^2j + 4ij - 1 \end{pmatrix}$$

Proof. Let $\Sigma_2 = \{c, d, \overline{c}, \overline{d}\}$. Since α and β are homomorphisms, we have that $\alpha(z_i^j) = \alpha(z_i)^j = c^i d^j \overline{c}^i$ and

$$\begin{aligned}\beta(\alpha(z_i^j)) &= \beta(c)^i\beta(d)^j\beta(\overline{c})^i \\ &= (rsr)^{2i}(rs)^{2j}(r^2sr^2)^{2i}\end{aligned}$$

Elementary matrix multiplication of $\beta \circ \alpha(z_i^j)$ reveals that

$$\begin{aligned}\beta \circ \alpha(z_i^j) &= (rsr)^{2i}(rs)^{2j}r^2(sr)^{2i}r \\ &= \{\pm I\}\begin{pmatrix} -8i^2j-4ij-1 & -8i^2j \\ 8i^2j+8ij+2j & 8i^2j+4ij-1 \end{pmatrix}\end{aligned}$$

We see that the size of the maximal element of the matrices $\beta(\alpha(z_i^j))$ is polynomial in i and j and thus size$(\beta(\alpha(z_i^j))) = O(i^2j)$. □

We need one final technical lemma concerning mapping $\beta \circ \alpha$.

Lemma 8. *For any nonempty reduced word $w \in \Sigma^+$, let $A = \beta \circ \alpha(w)$ where we ignore the sign of the matrix. Then $A_{11} \neq 0$ and $A_{12} \neq 0$.*

Proof. By Lemma 6, we have that $\beta \circ \alpha(w)$ has first letter r and last letter r in its reduced representation under $\mathrm{PSL}_2(\mathbb{Z})$. Thus we see that

$$\begin{aligned}A = RXR &= \begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix} \\ &= \begin{pmatrix} -d & c-d \\ b+d & b+d-a-c \end{pmatrix},\end{aligned}$$

for some matrix $X \in \mathrm{SL}_2(\mathbb{Z})$ (again we are ignoring the sign in this product).

We first prove $A_{12} \neq 0$. By the above formula, $A_{12} = 0$ implies $c - d = 0$. Since $\det(X) = 1$, $ad - bc = 1$ implies $c(a-b) = 1$, thus either $c = d = 1$ and $b = a - 1$, or else $c = d = -1$ and $b = a + 1$. Therefore either:

$$A = R\begin{pmatrix} a & a-1 \\ 1 & 1 \end{pmatrix}R \quad \text{or} \quad A = R\begin{pmatrix} a & a+1 \\ -1 & -1 \end{pmatrix}R.$$

It is not hard to see that matrices X of this form have factorizations $(SR)^xR$ or $(RRS)^xR$ for some $x > 0$. This holds since:

$$(SR)^kR = (-1)^k\begin{pmatrix} k & k-1 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad (RRS)^kR = (-1)^k\begin{pmatrix} -k & -(k+1) \\ 1 & 1 \end{pmatrix}$$

and under $\mathrm{PSL}_2(\mathbb{Z})$ we factor out $-I$ and can therefore ignore the sign. Thus, we see that $A = R(SR)^xRR$ or $A = R(RRS)^xRR$ for some $x > 0$. However, these are not reduced representations (since they have R^3 on the left or right) and since these reduced factorizations are unique, this contradicts Lemma 6 since under $\mathrm{PSL}_2(\mathbb{Z})$ A would start with s.

We now prove $A_{11} \neq 0$. Clearly $A_{11} = 0$ implies $d = 0$. Since $\det(A) = 1$, then $b = \pm 1$ and $c = \mp 1$. Thus $A = \begin{pmatrix} 0 & \pm 1 \\ \mp 1 & x \end{pmatrix}$ for some $x \in \mathbb{Z}$. Now, such A have reduced factorization (up to sign) of $(RS)^aR$ where $a \geq 0$ or $(SRR)^aS$ where $a > 0$ which is easy to check via straight forward matrix calculations. However, Lemma 6 shows that A should end with 'r' in its reduced representation under

$\mathrm{PSL}_2(\mathbb{Z})$, thus it must be of the form $(RS)^aR$. It is not hard to see however that $(rs)^ar \notin \{\beta(w) \mid w \in \Sigma_2^+\}$ for any $a \geq 0$. This follows because $\beta(w) = (rs)^ar = \beta(d)(rs)^{a-2}r = \ldots = \beta(d)^{a/2}r$ which cannot be further factorized giving a contradiction. □

Combining this information, we now have the following four essential properties:

i) $\beta \circ \alpha : \Sigma^* \hookrightarrow \mathrm{PSL}_2(\mathbb{Z})$ is a monomorphism by Lemma 4 and Lemma 5.

ii) For all nonempty reduced $w \in \Sigma^+$, $\beta \circ \alpha(w)$ has reduced representation $rw'r$ over $\mathrm{PSL}_2(\mathbb{Z}) \cong \langle s, r | s^2 = r^3 = 1 \rangle$ for some $w' \in \{s, r\}^*$. This follows from Lemma 6.

iii) For any $z_i \in \Sigma$, the size of matrices in $\beta \circ \alpha(z_i^j)$ in terms of the number of bits to represent it is logarithmic in terms of i and j. This follows from Lemma 7.

iv) For any nonempty reduced word $w \in \Sigma^+$, the upper left and upper right entries of matrices $\beta \circ \alpha(w)$ are nonzero by Lemma 8.

We are now ready to prove the main result of this section.

Theorem 1. *The mortality problem for matrices in $\mathbb{Z}^{2\times 2}$ is NP-hard.*

Proof. We adapt the proof from [3] which shows that the identity problem in $\mathbb{Z}^{2\times 2}$ is NP-hard. The proof in [3] essentially consists of two parts. First, an encoding is shown from the subset sum problem to a problem on words - given a finite set of words, can they be combined in such a way as to reach the identity (or empty) word. The number of letters in these words is exponential in the representation size of the subset sum problem instance however.

Therefore the second half of the proof shows a mapping from this set of words into $\mathbb{Z}^{2\times 2}$ such that the matrix representation of the words has size polynomial in the subset sum problem instance. The set of matrices generate a semigroup containing the identity matrix if and only if the subset sum problem has a solution, thus the identity problem is NP-hard for 2×2 matrix semigroups.

For our purposes of the mortality problem, we shall use the identical first part of the proof to give a set of words W encoding a subset sum problem instance. We shall then define a matrix P and use mapping $\beta \circ \alpha : \Sigma^* \hookrightarrow \mathrm{PSL}_2(\mathbb{Z})$ and its properties to encode the set of words W in such a way that the zero matrix is in the semigroup generated by a certain set of matrices if and only if there exists a solution to the subset sum problem.

Let $\Sigma = \{1, 2, \ldots, 2k+2, \overline{1}, \overline{2}, \ldots, \overline{(2k+2)}, a, b, \overline{a}, \overline{b}\}$ be an alphabet. The subset sum instance is given by $S = \{s_1, s_2, \ldots, s_k\}$ and value x - thus the problem is: does there exist a subset of S whose sum is x? We now define set of words:

$$W = \begin{array}{ll} \{1\cdot a^{s_1}\cdot\overline{2}, & 1\cdot\varepsilon\cdot\overline{2},\\ 2\cdot a^{s_2}\cdot\overline{3}, & 2\cdot\varepsilon\cdot\overline{3},\\ \vdots & \vdots\\ k\cdot a^{s_k}\cdot\overline{(k+1)}, & k\cdot\varepsilon\cdot\overline{(k+1)},\\ (k+1)\cdot\overline{a}^{x}\cdot\overline{(k+2)}, & \\ (k+2)\cdot b^{s_1}\cdot\overline{(k+3)}, & (k+2)\cdot\varepsilon\cdot\overline{(k+3)},\\ (k+3)\cdot b^{s_2}\cdot\overline{(k+4)}, & (k+3)\cdot\varepsilon\cdot\overline{(k+4)},\\ \vdots & \vdots\\ (2k+1)\cdot b^{s_k}\cdot\overline{(2k+2)}, & (2k+1)\cdot\varepsilon\cdot\overline{(2k+2)},\\ (2k+2)\cdot\overline{b}^{x}\cdot\overline{1}\} \subseteq \varSigma^* & \end{array}$$

It was proven in [3] that $\varepsilon \in W^+$ if and only if the subset sum instance S has a solution. We shall not repeat the details here, suffice it to say that letters $\{1,\ldots,2k+2,\overline{1},\ldots,\overline{2k+2}\}$ act as 'border letters' which enforce a particular ordering on any possible words reducing to give ε. In any such word $w' = w_1w_2\cdots w_l \in W^+$ such that $r(w') = \varepsilon$, we can assume by [3] that w_1 is from the first row of W (above), w_2 is from the second row etc. and that $w_l = (2k+2)\cdot\overline{b}^{x}\cdot\overline{1}$, thus $l = 2k+2$.

We now use mapping $\beta\circ\alpha : \varSigma^* \hookrightarrow \mathrm{PSL}_2(\mathbb{Z})$ applied to set of words W. We earlier defined this function on a different alphabet but the same analysis holds.

Since any element a in $\mathrm{PSL}_2(\mathbb{Z})$ already is a set $\{A, -A\}$ of two matrices in $\mathrm{SL}_2(Z)$ we set M in a way that it will actually contain $2|W|$ matrices. It does not alter this construction, since it does not matter if you select A or $-A$ in the product. Specifically we have

$$M = \{\beta\circ\alpha(w) | w \in W\} \subseteq \mathbb{Z}^{2\times 2},$$

thus $|M| = 2|W|$. Note that each $w \in W$ contains two border letters and possibly a power of a single letter from $\varSigma$. By Lemma 7, this implies that the size of the matrices in M (i.e. number of bits required to represent them) is polynomial in the number of bits to represent the subset sum problem instance.

Our next steps are to introduce a new matrix P and to modify one of the matrices in M by right multiplying by matrix S which will make it possible to reach the zero matrix if and only if $I \in \langle M\rangle$.

Let $P = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ and define $M' = M \cup \{P\}$. For any matrix $X \in \mathbb{Z}^{2\times 2}$ we have that $(PXP)_{11} = X_{11}$ and $(PXP)_{12} = (PXP)_{21} = (PXP)_{22} = 0$. Since for any reduced word $w \in W^+$, we have that $\beta\circ\alpha(w)_{11} \neq 0$ by Lemma 8, and all matrices in M are unimodular, then the zero matrix is not in $\langle M'\rangle$.

We now modify the matrix set M' one final time. Let Y be the matrix in M' corresponding to word $(2k+2)\cdot\overline{b}^{x}\cdot\overline{1} \in W$. We now form set $M'' = (M' \setminus \{Y\}) \cup \{YS\}$, i.e. we replace Y in M' by YS. Since all other matrices in M' have reduced factorizations of the form $RX'R$ for some $X' \in \{S,R\}^*$, the right multiplication of Y by S in this way does not allow any additional cancelation of elements. More formally, for any non-identity $X_1Y \in \langle M'\rangle$ and $X_2 \in \langle M'\rangle$ Lemma 6 shows that we have reduced representations $X_1Y = RX_1'R$ and $X_2 = RX_2'R$ for some $X_1' \in \{S,R\}^*$ and $X_2' \in \{S,R\}^*$. Therefore, under M'', since we replace Y by YS, then $X_1(YS)X_2 = RX_1'R \cdot S \cdot RX_2'R$ is also a reduced representation and no cancelation has occured by replacing Y with YS.

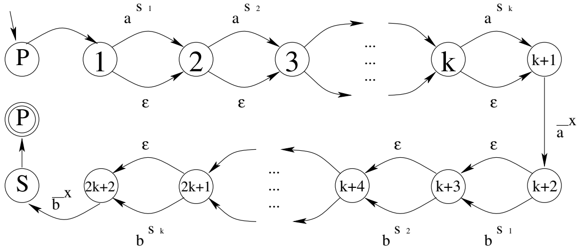


Fig. 1. The structure of a solution to the mortality problem

By Lemma 6, all non-identity matrices in M' have reduced (and unique) factorization RXR over $\{R,S\}$ for some $X \in \{S,R\}^*$. The only matrix in M' with a zero upper right corner (up to sign) is the identity matrix by Lemma 8. From the proof of NP-hardness of the identity problem in [3], any reduced word in W^+ equal to the empty word must be of the form $w'((2k+2)\cdot\overline{b}^x\cdot\overline{1})$ for some $w' \in W^+$, in other words the last word from W used must be $(2k+2)\cdot\overline{b}^x\cdot\overline{1}$. This word was represented by matrix Y under M', which we replaced with YS in M''. Thus, let $VY \in \langle M' \rangle$ be a product equal to the identity matrix. Therefore we see that $S = VYS \in \langle M'' \rangle$ and therefore $PVYSP = PSP$ is the zero matrix as required. This follows since

$$\begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} S = \begin{pmatrix} x_{12} & -x_{11} \\ x_{22} & -x_{21} \end{pmatrix} \tag{3}$$

The structure of such a product $PVYSP$ can be seen in Figure 1. To reach the zero matrix, we first use matrix P. We follow that by the matrix corresponding to either word $1 \cdot a^{s_1} \cdot \overline{2}$ or word $1 \cdot \varepsilon \cdot \overline{2}$, meaning we move from node 1 to node 2 and choose either a^{s_1} or else ε. This continues iteratively until we reach node $k+1$ at which point we move to $k+2$ with word $\overline{a}^x$. At this point, if the selected nonempty words are such that $a^{s_{j1}}a^{s_{j2}}\cdots a^{s_{jm}} \cdot \overline{a}^x = \varepsilon$ for some $1 \leq s_{j1} < s_{j2} < \ldots < s_{jm} \leq k$, then this corresponds to a correct solution to the subset sum problem instance. The same procedure holds between nodes $k+2$ and $2k+2$ (with a's replaced by b's) at which point we then use matrix S and the final matrix P. In the diagram, nodes 1 to $2k+1$ correspond to matrix V and nodes $2k+2$ and S correspond to matrices Y and S respectively. □

Since an upper bound for the decidability result in [12] is unknown, we now consider a lower bound on the minimum length solution to the Mortality Problem. Below we derive a lower bound on the minimum length solution to the Mortality Problem for a constructible set of instances, which is *exponential* in the number of matrices of the generator set and the maximal element of the matrices. This bound shows that the most obvious candidate for an NP algorithm, which is to guess the shortest sequence of matrices which multiply to give the zero matrix, does not work correctly since the certificate would have a length which is exponential in the size of the instance.

Theorem 2. *There exists a set of matrices $M = \{M_1, M_2, \ldots, M_n\} \subseteq \mathbb{Z}^{2\times 2}$ where the maximum element of any matrix in M is $O(n^2)$ such that $0 \in \langle M\rangle$ (where 0 here denote the zero matrix) and the minimal length product over M equal to 0 is of length 2^n, which is exponential in the number of matrices in the generator and the maximal element of any matrix in M.*

Proof. Let $\Sigma = \{1, 2, \ldots, 2n-1, \overline{1}, \overline{2}, \ldots, \overline{2n-1}\}$ be a group alphabet. It is proven in [3] that there exists a set of words $V = \{v_1, v_2, \ldots, v_{2n-2}\} \subseteq \Sigma^3$ such that there exists $w \in V^+$ where $r(w) = \varepsilon$ and $|w| = 2^n - 2$ and for all $w' \in V^+$ such that $|w'| < 2^n - 2$, then $r(w') \neq \varepsilon$. This results from an encoding of a deterministic finite automaton introduced in [1].

First, we encode set of words V into matrices. We apply monomorphism $\beta \circ \alpha : \Sigma^* \hookrightarrow \mathrm{PSL}_2(\mathbb{Z})$ to the set of words V to give

$$V' = \{\beta \circ \alpha(v) | v \in V\} \subseteq \mathrm{PSL}_2(\mathbb{Z})$$

From the encoding of the DFA used in [3], it is shown that for any $w \in V^+$ where $r(w) = \varepsilon$, we may assume that the last letter of w is v_{2n-2}. We proceed in a similar manner to that of the proof of Theorem 1. We form the set

$$V'' = (V' \setminus \{\beta \circ \alpha(v_{2n-2})\}) \cup \{\beta \circ \alpha(v_{2n-2})S\},$$

i.e. we right multiply the final matrix of set V' by matrix S. Finally, we define matrix $P \in \mathbb{Z}^{2\times 2}$ such that $P_{11} = 1$ and $P_{12} = P_{21} = P_{22}$ and let $V''' = V'' \cup \{P\}$.

For any $w \in V^+$ such that $r(w) = \varepsilon$, then $\beta \circ \alpha(w) = \pm I$ and thus $\beta \circ \alpha(w)S \in V'''$ with $|w| \geq 2^n - 2$. Clearly,

$$P(\beta \circ \alpha(w))SP = \beta \circ \alpha(w)_{12},$$

which equals 0 if and only if $r(w) = \varepsilon$ by Lemma 8 and the number of matrices used is 2^n. For all $X \in V'''$, then $X_{11} \neq 0$ by Lemma 8 and Equation (3). Thus only matrix $S \in V'''$ is such that $PSP = 0$ where 0 is here the zero matrix.

Finally we need to consider the representation size of V'''. Lemma 7 shows that for $|\Sigma| = 4n - 2$, we have that $\mathrm{size}(\beta \circ \alpha(x)) = O(n^2)$ for any $x \in \Sigma$. Since $|V'''| = 2n - 1$, then $\mathrm{size}(V''') = O(n^3)$ where size denotes the number of bits required to represent set V'''. □

References

1. Ang, T., Pighizzini, G., Rampersad, N., Shallit, J.: Automata and reduced words in the free group. arXiv:0910.4555 (2009)
2. Bell, P.C., Potapov, I.: On the undecidability of the identity correspondence problem and its applications for word and matrix semigroups. International Journal of Foundations of Computer Science 21(6), 963–978 (2010)
3. Bell, P.C., Potapov, I.: On the computational complexity of matrix semigroup problems. Fundamenta Informaticae 116(1-4), 1–13 (2012)
4. Birget, J.-C., Margolis, S.: Two-letter group codes that preserve aperiodicity of inverse finite automata. Semigroup Forum 76(1), 159–168 (2008)
5. Blondel, V., Tsitsiklis, J.: When is a pair of matrices mortal? Information Processing Letters 63, 283–286 (1997)
6. Blondel, V., Tsitsiklis, J.: The boundedness of all products of a pair of matrices is undecidable. Systems and Control Letters 41(2), 135–140 (2000)
7. Bournez, O., Branicky, M.: On the mortality problem for matrices of low dimensions. Theory of Computing Systems 35(4), 433–448 (2002)
8. Cai, J.-Y., Liu, Z.: The bounded membership problem of the monoid $\mathrm{SL}_2(N)$. Mathematical Systems Theory 29(6), 573–587 (1996)
9. Choffrut, C., Karhumäki, J.: Some decision problems on integer matrices. Informatics and Applications 39, 125–131 (2005)
10. Krom, M.: An unsolvable problem with products of matrices. Mathematical Systems Theory 14, 335–337 (1981)
11. Miller, M.A.: Mortality for sets of 2x2 matrices. Mathematics Magazine 67(3), 210–213 (1994)
12. Nuccio, C., Rodaro, E.: Mortality Problem for 2 x 2 Integer Matrices. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 400–405. Springer, Heidelberg (2008)
13. Paterson, M.S.: Unsolvability in 3 x 3 matrices. Studies in Applied Mathematics 49, 105–107 (1970)
14. Rankin, R.: Modular Forms and Functions. Cambridge University Press, Cambridge (1977)

Solving Counter Parity Games

Dietmar Berwanger[1], Łukasz Kaiser[2], and Simon Leßenich[1,3,⋆]

[1] LSV, CNRS & ENS Cachan, France
[2] LIAFA, CNRS & Université Paris Diderot – Paris 7, France
[3] Mathematische Grundlagen der Informatik, RWTH Aachen, Germany

Abstract. We study a class of parity games equipped with counters that evolve according to arbitrary non-negative affine functions. These games capture several cost models for dynamic systems from the literature. We present an elementary algorithm for computing the exact value of a counter parity game, which both generalizes previous results and improves their complexity. To this end, we introduce a class of ω-regular games with imperfect information and imperfect recall, solve them using automata-based techniques, and prove a correspondence between finite-memory strategies in such games and strategies in counter parity games.

1 Introduction

Games with ω-regular winning conditions, and especially parity games, are a fundamental model for program verification and synthesis [12]. Such winning conditions allow to express reachability, safety, and liveness properties. However, when specifying, for instance, that for each request Q_i there will be finally a response R_i, one is often interested not only in the existence of a response R_i – a *qualitative* property, but also that the response will occur in at most k seconds after the request – a *quantitative* constraint.

Quantitative questions about reactive systems have been approached in several ways. One possibility is to extend a temporal logic with new, quantitative operators as, for instance, the "prompt" operator for LTL proposed in [11]. While the existence of a response R_i is formulated in LTL by $\mathsf{F}R_i$, the Prompt-LTL formula $\mathsf{F_p}R_i$ expresses that the waiting time is bounded. Realizability for this logic was solved in [11] and optimal bounds on the waiting time for Prompt-LTL formulas were established in [14]. Another possibility is to consider formulas which evaluate to numbers rather than truth values. The quantitative version of CTL with discounts studied in [6], and the quantitative μ-calculus investigated in [7] follow this direction.

Both model-checking and realizability problems for most of these logics are reduced to solving games with additional quantitative features. Several classes of such games have therefore been investigated [3,4,5,9], to provide better algorithms for existing logics and to suggest new formalisms with good algorithmic properties. One relevant example is the synthesis of optimal strategies in

[⋆] This author was supported by the ESF Research Networking Programme GAMES and the DFG Research Training Group 1298 (AlgoSyn).

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 160–171, 2012.

request-response games [9]. Here, the problem of minimizing the waiting time for a response is investigated, considering different ways to accumulate waiting costs. In one model, the penalty for waiting k many steps is k^2, to discourage long waiting times. This illustrates that cost functions that depend only linearly on the time may not be sufficient for certain applications.

In this work, we introduce a model of counter parity games in which counters are updated by arbitrary non-negative affine transformations along the moves of a play. When a play ends, the counters are used to determine the payoff, whereas on infinite plays, a parity condition is applied. For example, the time since the last request Q_i can be stored in the counter c_i, which will be reset every time the response R_i arrives. The maximum value reached by c_i is then the maximal waiting time for R_i. Similarly, one can express arbitrary Prompt-LTL conditions. It is also possible to have another counter, d_i, which will increase *by* c_i every time the request Q_i is active, and will also be reset on R_i. Note that this is an affine update and that $c_i + 2d_i$ stores the waiting time squared, and thus allows to simulate the cost model from [9]. Affine functions also allow to swap counters and multiply them by constants, which can be used, e.g., to model process migration and pricing. This could be used to extend the scope of formal analysis of online algorithms [1]. Moreover, counter parity games are a strict generalization of counter-reset games used in [8] to approximate the quantitative μ-calculus over a class of hybrid systems. For model-checking this logic, only a non-elementary algorithm was known so far [8] – our results both allow for a more general class of games and provide better complexity bounds. In [10], counter parity games are used for proving decidability of the counting μ-calculus, an extension of the quantitative μ-calculus to structured transition systems, such as graphs generated by regular tree grammars or by pushdown automata.

2 Counter Parity Games

To define counter parity games, let us fix a natural number k of counters and let $\mathcal{F}_k$ be the set of k-dimensional affine functions f over non-negative integers:

$$f : \mathbb{N}^k \to \mathbb{N}^k, \quad f(c) = A \cdot c + B$$

for some matrix $A \in \mathbb{N}^{k \times k}$ and some vector $B \in \mathbb{N}^k$. Note that we only consider functions $\mathbb{N}^k \to \mathbb{N}^k$, thus the coefficients are also assumed to be non-negative.

A *counter parity game* $\mathcal{G} = (V, V_{\max}, V_{\min}, E, \Omega, \lambda)$ with k counters is played by two players, Maximizer and Minimizer, on a directed graph (V, E). The vertex set is partitioned into vertices $V_{\max}$ of Maximizer and vertices $V_{\min}$ of Minimizer. Vertices are colored by the priority function $\Omega : V \to \{0, \dots, d-1\}$, edges are labeled by affine functions, i.e., $E \subseteq V \times \mathcal{F}_k \times V$, and terminal vertices $T = \{v \mid vE = \emptyset\}$ are labeled by $\lambda : T \to \{+, -\} \times \{0, \dots, k-1\}$.

The k counters are represented by a vector $c \in \mathbb{N}^k$ of k natural numbers. We write c_i for the i-th component of c, i.e., the i-th counter. At the beginning of a play, all counters are 0, thus $c = 0^k$. Throughout a play, counters are updated according to the labels of the edges: if the current value of the counter vector is c and an edge (u, f, v) is taken, then the new value is $f(c)$. Maximizer moves at positions $V_{\max}$, while Minimizer moves at $V_{\min}$.

A *play* $\pi = v_0 f_0 v_1 f_1 v_2 \ldots$ is a sequence of vertices and edge labels such that, for each $i \geq 0$, $(v_i, f_i, v_{i+1}) \in E$. For infinite plays π, the payoff $p(\pi)$ is determined by the parity condition given by Ω: it is $-\infty$ if the minimal priority seen infinitely often in $\Omega(v_0)\Omega(v_1)\ldots$ is odd, and ∞ if it is even. Finite plays π end at a terminal vertex t and $p(\pi)$ is determined by $\lambda(t)$ and the current counters: it is $s\, c_i$ if $\lambda(t) = (s, i)$ and the current vector is c. The objective of Maximizer is to maximize the payoff, whereas Minimizer seeks to minimize it.

A *strategy* of Maximizer is a function $f : (V\mathcal{F}_k)^* V_{\max} \to \mathcal{F}_k \times V$, such that, for each prefix π of a play, if $f(\pi v) = (f, w)$ then $(v, f, w) \in E$; analogously, we define strategies of Minimizer $f : (V\mathcal{F}_k)^* V_{\min} \to \mathcal{F}_k \times V$. We say that f uses *memory* M if there exists a $m_0 \in M$, a function $\text{update} : M \times \mathcal{F}_k \times V \to M$, and a function $f_M : M \times V \to \mathcal{F}_k \times V$ such that $f(v_0 f_0 v_1 \ldots f_{n-1} v_n) = f_M(\text{update}^*(v_0 f_0 v_1 \ldots f_{n-1} v_n, m_0), v_n)$, where update^* is defined inductively by $\text{update}^*(\varepsilon, m) = m$ and

$$\text{update}^*(v_0 f_0 v_1 \ldots f_k v_{k+1}, m) = \text{update}(\text{update}^*(v_0 f_0 v_1 \ldots v_k, m), f_k, v_{k+1}).$$

The *size* of the memory is $|M|$ and a *finite-memory strategy* is one that uses a finite memory M.

Strategies can be identified with labelings of the infinite tree $\mathcal{T}(\mathcal{G}, v_0)$ obtained by unfolding the arena of $\mathcal{G}$ from v_0. This allows to speak about *regular* sets of strategies, i.e., sets which are recognized by non-deterministic parity tree automata over $\mathcal{T}(\mathcal{G}, v_0)$. We assume that the reader is familiar with this standard way of identifying strategies with labelings of the infinite tree. We will also use algorithms for alternating automata on infinite trees, which are more precisely recalled in the technical report [2].

A counter parity game $\mathcal{G}$ is *determined* if the supremum of the payoffs that Maximizer can achieve coincides with the infimum of the payoffs that the Minimizer cannot avoid, that is, if

$$\sup_{f \in \Sigma_{\max}} \inf_{g \in \Sigma_{\min}} p(\alpha_{f,g}(v)) = \inf_{g \in \Sigma_{\min}} \sup_{f \in \Sigma_{\max}} p(\alpha_{f,g}(v)) =: \operatorname{val} \mathcal{G}(v),$$

where $\Sigma_{\max}$ and $\Sigma_{\min}$ are the sets of all strategies of Maximizer and Minimizer, and $\alpha_{f,g}(v)$ is the unique play consistent with both f and g.

As counter parity games are a special case of quantitative parity games on infinite arenas (we can encode counter values in the vertices and adjust the edges accordingly), and it was shown in [7] that quantitative parity games are determined on arenas of arbitrary size, we obtain the following corollary.

Proposition 1 ([7]). *Every counter parity game $\mathcal{G}$ is determined: for each vertex v the value* $\operatorname{val} \mathcal{G}(v)$ *exists.*

However, these results do not imply that the value of a counter parity game can actually be computed, nor do they give any insight into the structure of strategies in the game. Our main technical result, stated below, identifies good regular over-approximations for the strategies of Minimizer. Let us denote by 2EXP(f) the family of functions $2^{2^{\mathcal{O}(f)}}$ and, for a counter parity game $\mathcal{G}$ and a strategy g of Minimizer, let us write $\mathrm{val}_g\,\mathcal{G}(v) = \sup_{f \in \Sigma_{\max}} p(\alpha_{f,g}(v))$ for the supremum of all payoffs Maximizer can get when playing against g.

Theorem 2. *Let $\mathcal{G} = (V, V_{\max}, V_{\min}, E, \Omega, \lambda)$ be a counter parity game with k counters, and let $v_0 \in V$. One can compute a constant $m = 2\mathrm{EXP}(k)$ and a regular set Σ of strategies of Minimizer, recognized by a non-deterministic parity tree automaton of size $2\mathrm{EXP}((|V|+km)^3)$ and with polynomial index, such that:*

- *for every strategy $g \notin \Sigma$, $\mathrm{val}_g\,\mathcal{G}(v_0) = \infty$, and*
- *for every memory M strategy $g \in \Sigma$, $\mathrm{val}_g\,\mathcal{G}(v_0) < 2\mathrm{EXP}(|M|^2 \cdot (|V| \cdot 2^k)^4)$.*

Note that the set Σ is only an over-approximation of the strategies of the Minimizer which guarantee a bounded payoff – only finite-memory strategies from Σ have this property. The following example illustrates why this is a crucial constraint. It also shows that the exact set of strategies which guarantee a bounded payoff is not regular, which is why we compute an over-approximation.

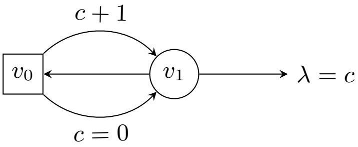


Fig. 1. Counter Parity Game with Value 0

Example 3. Let $\mathcal{G}$ be the 1-counter parity game depicted in Figure 1. Minimizer's vertices are drawn as squares and Maximizer's as circles. The game proceeds as follows: Minimizer can either increment the single counter c or reset it, and then Maximizer can decide to continue or to exit and take the current value of c. The priorities in the game are all odd, thus Maximizer must exit at some point.

Clearly, Minimizer can reset c at each step and thus $\mathrm{val}\,\mathcal{G}(v_0) = 0$. But which strategies g of Minimizer guarantee $\mathrm{val}_g\,\mathcal{G}(v_0) < \infty$? Of course, these are exactly the strategies that do not allow c to grow beyond some bound B, i.e., alternate the two possible moves $(c = 0)$ and $(c+1)$ according to the pattern

$$(c+1)^{n_1}(c=0)^+(c+1)^{n_2}(c=0)^+(c+1)^{n_3}(c=0)^+\cdots,$$

such that, for some $B \in \mathbb{N}$, $n_i < B$ for all i. But this set, i.e., the set of all strategies g which guarantee that $\mathrm{val}_g\,\mathcal{G}(v_0) < \infty$, is not regular.

To over-approximate the strategies which guarantee a bounded payoff, we will put in Σ all strategies that take the reset move $(c = 0)$ infinitely often. Note that this disregards the restriction on the number of $(c+1)$-moves between resets. All *finite-memory* strategies from Σ indeed guarantee a bounded value, because the number of increments between resets cannot exceed the memory size. However, consider the infinite memory strategy g_* which plays according to the pattern above with $n_i = i$. This strategy is in Σ, but $\mathrm{val}_{g_*} \mathcal{G}(v_0) = \infty$.

We prove Theorem 2 in the following sections. Let us first state that it can be used to decide boundedness and compute the value of counter parity games.

Corollary 4. *Given a finite counter parity game $\mathcal{G}$ with initial vertex v, one can decide whether $\mathrm{val}\,\mathcal{G}(v) = \infty$ in* 2EXPTIME*, if the number of counters is fixed, and in* 4EXPTIME *otherwise. The value $\mathrm{val}\,\mathcal{G}(v)$ can be computed exactly in* 4EXPTIME *if the number of counters is fixed and in* 6EXPTIME *otherwise.*

3 Marks for Counter Updates

In this section, we make the first step towards proving Theorem 2 and introduce *marks* for counter update functions. Marks are an abstraction and allow to determine whether a counter increased with respect to other counters or not.

Notation. When referring to a sequence s, we write $s[i]$ for the i-th element of s. We always count from 0, i.e., $s[0]$ is the first element of s. For a set $I \subseteq \mathbb{N}$ of indices, we write $s|^I$ to denote the sub-sequence of s consisting only of the elements with indices in I. We refer to a sequence of fixed finite length as a vector. For a vector s, we write $s^{>0}$ to denote the vector t with $t[i] := 1$ if $s[i] > 0$ and $t[i] := 0$ otherwise. Finally, we write $[n]$ to denote the set $\{0, \ldots, n-1\}$.

Let k be the dimension of the counter vector $c \in \mathbb{N}^k$. We consider all counter update functions $f : \mathbb{N}^k \to \mathbb{N}^k$ that admit marking in the following sense.

A *mark* is a mapping $m : \{0,1\}^k \times [k] \to \{\bot\} \uplus [k] \uplus \mathcal{P}([k])$. A function $f : \mathbb{N}^k \to \mathbb{N}^k$ *has mark* m if the following hold for all $c \in \mathbb{N}^k, i \in [k]$.

(i) If $m(c^{>0}, i) = \bot$ then $f(c)[i] = 0$.
(ii) If $m(c^{>0}, i) = j \in [k]$ then $f(c)[i] = c_j$.
(iii) If $m(c^{>0}, i) = D \in \mathcal{P}([k])$ and $D \neq \emptyset$ then $f(c)[i] > \max_{j \in D} c_j$.
(iv) If $m(c^{>0}, i) = \emptyset$ then $f(d)[i] = C > 0$ is constant for all d with $d^{>0} = c^{>0}$.
(v) $f(c)[i]$ depends only on the counters from $m(c^{>0}, i) = D$,
i.e., there exists a function f'_i such that $f(c)[i] = f'_i(c|^D)$.

Note that (iv) could be seen as special case of (v), but we distinguish whether the constant is 0, as in (i), or not. Intuitively, a mark determines, depending on which counters are 0 and which are not, whether the result will be 0, always stay equal to another counter, or increase over other counters.

In particular, if $m(d, i) = D$ then, after applying the counter update function, the (value of) counter i will be strictly greater than each of the counters from D. We write $m^{\geq}(d, i)$ for the set D of counters such that the value of c_i after the update will be greater or equal to the values of the counters in D, i.e., $m^{\geq}(d, i) := \emptyset$ if $m(d, i) = \bot$, $m^{\geq}(d, i) := \{l\}$ if $m(d, i) = l$, and $m^{\geq}(d, i) := D$ if $m(d, i) = D \in \mathcal{P}([k])$. Additionally, we write $m^{>0}(c^{>0})$ for the vector $d^{>0}$ if d

results from the application of a function f with mark m to the vector c. Observe that $f(c)[i] = 0$ if, and only if, $m(c^{>0}, i) = \bot$ or $m(c^{>0}, i) = l$ and $c_l = 0$, and thus $m^{>0}(c^{>0}) = f(c)^{>0}$ is computable from $c^{>0}$ and m.

Example 5. Consider two counters c_0, c_1 and the update function f assigning $c_0 + c_1$ to c_0 and $2 \cdot c_0$ to c_1. This function has the following mark m: $m(0, 0, i) = \bot$, $m(0, 1, 0) = 1$ as $c_0 + c_1 = c_1$ if $c_0 = 0$, and $m(1, 0, 0) = 0$ analogously; $m(0, 1, 1) = \bot$ as $2 \cdot 0 = 0$, but $m(1, 0, 1) = m(1, 1, 1) = \{0\}$ as $2 \cdot c_0 > c_0$ for $c_0 > 0$. Finally, $m(1, 1, 0) = \{0, 1\}$ as c_0 exceeds both counters in this case.

Not only affine functions can be marked. For example, the function which updates c_i to $\max(c_j, c_l) + 1$ also has a mark, and thus our results also hold for such functions (in fact, a mark D expresses a lower bound of $\max D + 1$). Note also that not all functions admit a marking. For example, if we updated c_1 to $c_0 \cdot c_1$ above, we would not be able to assign a mark. In particular, $m(1, 1, i)$ is not definable, because whether the counter increases or stays unchanged depends on whether $c_i > 1$ and not just on whether $c_i > 0$. The methods we present generalize to more involved markings, but we do not introduce them here as we are interested in one class of functions, for which the above marks suffice.

Lemma 6. *Let $f : \mathbb{N}^k \to \mathbb{N}^k$ be affine. There exists a mark m_f for f.*

Another important property of marks (see [2] for the proofs) is that, when functions are composed, their marks can be composed as well.

Lemma 7. *Let f_1 and f_2 be counter update functions with marks m_1 and m_2. A mark $m = m_1 \circ m_2$ for $f(c) = f_2(f_1(c))$ can be computed from m_1 and m_2.*

Let us denote by $\mathcal{M}$ the set of all marks, which is finite, by definition. For a fixed number k of counters, $|\mathcal{M}| \leq \left(2^k + k + 1\right)^{2^{k+\log k}} = 2\mathrm{EXP}(k)$. Moreover, by the above lemma, the composition $\circ$ induces a computable finite semigroup structure on $\mathcal{M}$. It follows that languages of sequences of marks with definable properties are regular. For example, the language of all sequences $m_0 m_1 \dots m_n \in \mathcal{M}^*$ such that $m = m_1 \circ \cdots \circ m_n$ satisfies, for a fixed C, i and d, that $C \subseteq m^{\geq}(d, i)$, is regular. This means that, for a fixed set of counters C and starting information about which counter is 0, we can determine in a regular way whether c_i will be at least as large as some counter from C.

To access marks, we extend counter parity games by the appropriate marking. Let $\mathcal{G}$ be a counter parity game with k counters. The *marked counter parity game* $\mathcal{G}_m = (V_m, V'_{\max}, V'_{\min}, E_m, \Omega_m, \lambda_m)$ is a game with k counters defined as follows.

- $V_m := V \times \{0, 1\}^k$ (storing which counters are greater than 0).
- $V'_{\max} = \{(v, x) \in V_m \mid v \in V_{\max}\}$, $V'_{\min} = V_m \setminus V'_{\max}$.
- $E_m \subseteq V_m \times (\mathcal{F}_k \times \mathcal{M}) \times V_m$ stores the marks and updates the $c^{>0}$-vectors:

$$E_m := \{((u, x), (f, m_f), (v, m_f^{>0}(x))) \mid (u, f, v) \in E,\ x \in \{0, 1\}^k\}.$$

- $\Omega_m(v, x) = \Omega(v)$ and $\lambda_m(v, x) = \lambda(v)$.

4 Second-Life Games

In this section, we introduce the class of second-life games with imperfect information and a specific kind of imperfect recall, which are essential for the construction in the next section. The construction of a second-life game starts with a game graph $\mathcal{G}$ with perfect information for two players, Player 0 and Player 1. The second-life game arena consists of several copies of this graph. Plays begin in the main instance of $\mathcal{G}$, which we call *first life*, and proceed as usual by moving a token along the edges of the graph. However, when Player 1 is in turn to move, he may switch to a copy of $\mathcal{G}$, a *second life*, without informing Player 0. If a terminal position is reached in the first life, the play simply ends. In contrast, if this happens in a second life, the play returns to the first life, and Player 0 forgets the part of the history spent in the second life. This part of the history is nevertheless relevant for the winning condition.

Let $\mathcal{G} = (V, V_0, V_1, E)$ be a game arena with $E \subseteq V \times A \times V$ for a set A of actions, and let $T = \{t \in V : tE = \emptyset\}$ denote the set of terminal vertices in $\mathcal{G}$. The *second-life game* $\mathcal{S}(\mathcal{G}, W)$ is a game with the set of actions

$$A_{\|} := A \cup \{\mathsf{Return}\} \cup \{\mathsf{Call}(a) \mid a \in A\}$$

and over the arena (V', V_0', V_1', E'), with

$$\begin{aligned} V' &:= V \cup (V \times V); \\ V_0' &:= V_0 \cup \{(u, v) \mid u \in V_0\} \cup \{(t, v) \mid t \in T, v \in V\}, \quad \text{and} \quad V_1' := V' \setminus V_0'; \\ E' &:= E \cup \{((u, v), a, (u', v)) \mid (u, a, u') \in E\} \\ &\quad \cup \{(u, \mathsf{Call}(a), (v, v)) \mid u \in V_1, (u, a, v) \in E\} && \text{(CALL)} \\ &\quad \cup \{((t, v), \mathsf{Return}, v) \mid t \in T\} && \text{(RETURN)} \end{aligned}$$

Winning conditions for second-life games have the form $W \subseteq A_{\|}^{\omega}$.

A play α is a – possibly infinite – alternating sequence of vertices and actions, $\alpha = v_0 a_0 v_1 a_1 v_2 \cdots$, such that $(v_i, a_i, v_{i+1}) \in E'$, for any index i. A finite play is one that ends at a terminal vertex. Every finite play is winning for Player 0; an infinite play is winning for Player 0 if, and only if, its action trace belongs to W.

Notice that all the moves of the arena $\mathcal{G}$ are available in the second-life game, regardless of whether the play is in the first or in a second-life copy. Additionally, when Player 1 moves at a vertex of the first-life copy $\mathcal{G}$, he can choose to switch to a second-life copy via a $\mathsf{Call}(\cdot)$ action.

The intended information structure of second-life games is captured by a constraint on strategies of Player 0. We postulate that Player 0 is not informed about whether the current vertex is in the main copy or in some other component. Furthermore, after any Call-Return sequence, Player 0 forgets the part of the play between Call and Return.

Here, and in the following, a Call-Return *sequence* is a sequence of the form

$$u \cdot \mathsf{Call}(a) \cdot (v, v) \cdot a_1 \cdot (v_1, v) \cdots (t, v) \cdot \mathsf{Return} \cdot v.$$

For any finite path π starting at a vertex v in the main copy, we define the path $\hat{\pi}$ obtained by replacing every Call-Return sequence $u \cdot \mathsf{Call}(a) \cdots \mathsf{Return} \cdot v$ by $u \cdot a \cdot v$, then replacing the remaining last $\mathsf{Call}(a)$ by a (if such a last Call exists), and finally projecting every occurring (u, v) to u.

Now, strategies of Player 0 are functions $f : (V' A_{||})^* V_0' \rightarrow A \times V$ such that, for every π ending in a vertex of Player 0 we have $f(\pi) = f(\hat{\pi})$. Thus, Player 0's strategies respect the information constraint described above. Strategies of Player 1 are not restricted in any way.

Notice that, for every play α, the sequence $\hat{\alpha}$ corresponds to a play in the main copy. However, W is given over $A_{||}$. Nonetheless, Player 0 has no information about whether he is moving in one of the second-life components or in the main copy, and immediately after noticing that the play continues after a terminal (which means it must have been in a second-life component), he forgets this and all that happened in the component. Accordingly, any strategy of Player 0 can be viewed as a strategy over the vertex set V with actions A, i.e., as a strategy of Player 0 for the arena $\mathcal{G}$.

Our main result on second-life games, proved using automata techniques (c.f. [2]), states that the set of winning strategies of Player 0 is regular and an automaton recognizing it can be constructed effectively.

Theorem 8. *Let $\mathcal{G}$ be an arena with positions V and W a regular winning condition recognized by a deterministic parity automaton $\mathcal{A}$. The set of winning strategies of Player 0 in the second-life game $\mathcal{S}(\mathcal{G}, W)$ can be recognized by a non-deterministic parity tree automaton of size at most $2^{\mathcal{O}(|\mathcal{A}|^9 + |V|^3)}$.*

5 The Unboundedness Game

In the next step, we consider a marked counter parity game and check whether its value is unbounded, i.e., ∞, or not. To do this, we transform the marked game into a second-life game, where Minimizer takes the role of Player 0.

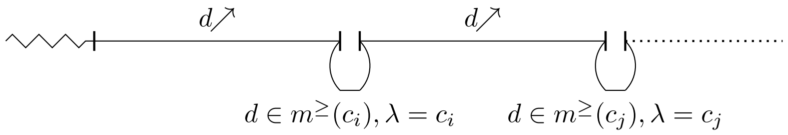
From the definition of the value of a counter parity game, there are two ways for the value to be ∞: Maximizer may have a winning strategy with respect to the parity condition, or a sequence $f_0, f_1, \cdots$ of strategies which ensure arbitrarily high payoffs. Via the reduction to second-life games, we combine the sequence of strategies for the latter situation into a single strategy. Intuitively, Maximizer will get the option to decide to try to reach a terminal position to "save" a payoff, and then continue increasing the counters. If in such a game Maximizer has a strategy to save higher and higher payoffs, or to win via the parity condition, this corresponds to a value of ∞. We exploit that marks form a finite semigroup to show that this can be formulated as a regular objective. Intuitively, the reason why we rely on second-life games with imperfect information and recall for Minimizer is that we need to avoid that Minimizer learns about whether Maximizer attempts to win by parity or by reaching arbitrarily high payoffs. If Minimizer had this information, he could adapt his strategy and neglect the other way of ensuring payoff ∞.

Let $\mathcal{G}_m$ be a marked counter parity game with arena $\mathcal{G}$ and terminal vertices T. The unboundedness game $\mathcal{G}^u$ is the second-life game $\mathcal{S}(\mathcal{G}_m, W)$ using $V_0 := V_{\min}$ and $V_1 := V_{\max}$ and with the winning condition W described below.

Recall that, if we remove all Call-Return sequences from a path in $\mathcal{G}^u$, we obtain a path in $\mathcal{G}_m$ that we call the main part. For better readability, we describe the winning condition in terms of both edge- and vertex-labels (functions/marks and priorities, respectively). Technically, this can be avoided by adding the color of the source vertex to the action label.

We describe the winning condition for Maximizer, i.e., Player 1, which is sufficient since regular languages are closed under complementation. Maximizer wins a play α if, and only if, the main copy is visited infinitely often, no terminal vertex inside the main copy is seen, and

– the main part satisfies the parity condition of $\mathcal{G}_m$, or
– there exists a counter d such that, from some point onwards, counter d is increased in the main part, then a Call is taken and a Return from a terminal where a payoff greater than d would be obtained in the original counter game, and after the Return this is repeated, ad infinitum.


$d\nearrow$
$d\nearrow$
$d \in m^{\geq}(c_i), \lambda = c_i$
$d \in m^{\geq}(c_j), \lambda = c_j$

By properties of marks, finite sequences of marks after which a counter d has been increased form a regular language $d\nearrow$. Also, finite sequences starting with a Call and ending with a Return from a vertex with $\lambda = c$ such that, for the sequence of marks in between, counter c is, at the end, greater than d at the beginning, form a regular language $c^{>d}$. Thus, the later part of W is the union of the main part satisfying the parity condition and the play being of the form $A_{\|}^* \cdot (d\nearrow \cdot c^{>d})^\omega$. This is ω-regular.

Let us calculate the size of the automaton for the above condition. To check the language $d\nearrow$, $|\mathcal{M}|$ states suffice for a deterministic finite word automaton (using composition on the marks), and the same holds for $c^{>d}$. Checking $d\nearrow \cdot c^{>d}$ can thus be done with $\mathcal{O}(|\mathcal{M}|)$ many states by a non-deterministic automaton. By considering Büchi acceptance with the same accepting states, we get a Büchi automaton for the language $(d\nearrow \cdot c^{>d})^\omega$ with $\mathcal{O}(|\mathcal{M}|)$ states. If we add a new initial state and take a copy of the automaton for $(d\nearrow \cdot c^{>d})^\omega$ for every $d < k$, we build a nondeterministic Büchi automaton of size $\mathcal{O}(k \cdot |\mathcal{M}|)$ which accepts a play if it is won via some counter. (It waits in the initial state until the actual d is correctly guessed and then moves to the respective copy.) For the parity part, we need an automaton of size $|\Omega(V)| < |V|$. Taking the union of the two, we get a non-deterministic parity automaton of size $\mathcal{O}(|V| + k|M|)$ and index $|V|$. After determinization, the deterministic parity automaton for W has size $2^{\mathcal{O}(|V|(|V|+k|\mathcal{M}|)\log(|V|+k|M|))} = 2^{\mathcal{O}((|V|+k|\mathcal{M}|)^3)}$.

Combining this with Theorem 8, we can conclude that the set of winning strategies of Minimizer in the unboundedness game can be recognized by a nondeterministic parity tree automaton of size

$$2^{\mathcal{O}\left(\left(2^{\mathcal{O}((|V|+k|\mathcal{M}|)^3)}\right)^9+|V|^3\right)} = 2\text{EXP}\left((|V|+k|\mathcal{M}|)^3\right). \tag{1}$$

What remains to be shown is the connection between the value of $\mathcal{G}$ and the existence of a winning strategy of Minimizer in $\mathcal{G}^u$, i.e., that the set of winning strategies of Minimizer in $\mathcal{G}^u$ satisfies the conditions from Theorem 2. We will use Ramsey's theorem for a finite path π in $\mathcal{G}^u$ or $\mathcal{G}$ played consistently with a strategy using memory of size K_0. We write π as a sequence of vertices and memory states with edges labeled with the corresponding marks:

$$\pi = (v_0, q_0) \xrightarrow{m_0} (v_1, q_1) \xrightarrow{m_1} (v_2, q_2) \cdots \xrightarrow{m_{n-2}} (v_{n-1}, q_{n-1}),$$

where each v_i is a vertex and each q_i is a memory state (and $q_{i+1} = \text{update}(q_i, v_i)$ according to the strategy). The path π induces a complete edge-colored undirected graph over $[n]$, in which an edge i, j is colored by (m, v_i, q_i, v_j, q_j), where m is the composition of the marks $m_i \circ m_{i+1} \circ \cdots \circ m_{j-1}$. Let l be the number of such colors for $\mathcal{G}^u$ and memory size K_0:

$$l = |\mathcal{M}| \cdot |V_u|^2 \cdot K_0^2 = |\mathcal{M}| \cdot K_0^2 \cdot (|V_m| \cdot |V_m|)^2 = |\mathcal{M}| \cdot K_0^2 \cdot (|V| \cdot 2^k)^4.$$

We write $\mathcal{R} = R(\underbrace{3, 3, \cdots, 3}_{l \text{ times}})$ for the Ramsey-number for 3-cliques with l colors. As $\mathcal{R} \leq 3l!$ [13], we get that $\mathcal{R} = 2^{\mathcal{O}(l)}$.

Recall that a mark m is idempotent if $m = m \circ m$. The following lemma (see [2]) is used in the next proof. To simplify notation, we write $i \mathrel{\underline{\underline{\in}}} m(c^{>0}, j)$ if $i \in m(c^{>0}, j)$ or $i = m(c^{>0}, j)$.

Lemma 9. *Let m be an idempotent mark. Then, for all initial values c, and all $i < k$: if $i \mathrel{\underline{\underline{\notin}}} m(c^{>0}, i)$, then i will not appear in any $m(c^{>0}, j)$.*

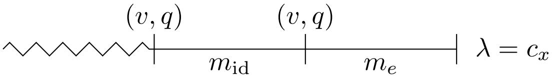
In the following, we show that, if g is a winning strategy of Minimizer in $\mathcal{G}^u$ with memory M, then $\text{val}_g\, \mathcal{G}(v)$ is bounded.

Proposition 10. *Let g be a strategy of Minimizer winning in $\mathcal{G}^u$ from v and using memory M. Then,* $\sup_{f \in \Sigma_{\max}} p(\alpha_{f,g}(v)) < 2\text{EXP}(M^2 \cdot (|V| \cdot 2^k)^4)$.

Proof. Let g be such an M-memory winning strategy. Consider the set of paths of length at most $\mathcal{R}$. We fix K as the maximal counter value plus 1 occurring anywhere on these paths when starting with initial counter values $c = (a, \cdots, a)$, where a is the maximal number occurring in any update function's matrices A or B. A rough upper bound can be computed as follows: after one application of any of the update functions, the maximal value is at most $a \cdot a \cdot k$ (the sum of all counters initialized with a, each weighted with a). After two steps, we get at most $k \cdot a \cdot k \cdot a \cdot a = k^2 a^{2+1}$. After $\mathcal{R}$ steps, we thus get $K \leq k^{\mathcal{R}} a^{\mathcal{R}+1} + 1 = 2^{\mathcal{O}(\mathcal{R})}$, and by the approximation of $\mathcal{R}$ above, $K \leq 2\text{EXP}(M^2 \cdot (|V| \cdot 2^k)^4)$.

Note that, because of imperfect information and imperfect recall, g can also be viewed as a strategy for $\mathcal{G}$. Consider thus, towards a contradiction, a play α in $\mathcal{G}$ that is consistent with g and that has a payoff $\geq K$. Let further β be the corresponding play in $\mathcal{G}^u$ in which Maximizer never takes a Call, i.e., β consists only of a main part. We distinguish two cases: if α is infinite, then so is β. Because g is a winning strategy for Minimizer in $\mathcal{G}^u$, β – and thus α – violates the parity condition. But then the payoff for α is $-\infty$, a contradiction. If α is finite but has a payoff $\geq K$, we first observe the following (proved in [2]).

Claim. Every play consistent with g and with payoff $\geq K$ has a suffix $(*)$:



with m_{id} idempotent, such that for some j with $j \leqq m_{\text{id}} \circ m_e(c_x)$: $j \in m_{\text{id}}(j)$.

By the above claim, it follows that α contains a cycle that can be repeated arbitrarily many times by Maximizer. As repeating the cycle increases the payoff, repeating the cycle, taking a Call towards the Return, then repeating the cycle and taking the Call again, and so on, is a witness for a win of Maximizer in $\mathcal{G}^u$. Because of imperfect information and imperfect recall, this witness is consistent with g, contradicting the assumption that g is winning for Minimizer. □

To prove the other item in Theorem 2, we show that Maximizer can achieve arbitrarily high payoffs against non-winning strategies of Minimizer in $\mathcal{G}^u$.

Proposition 11. *For every strategy g of Minimizer in $\mathcal{G}^u$ that is not winning from v,* $\text{val}_g\, \mathcal{G}(v) = \infty$*.*

Proof. Let g be an arbitrary strategy of Minimizer that is not winning from v in $\mathcal{G}^u$. This means that there exists a consistent play $\alpha(g)$ won by Maximizer. Note that $\mathcal{G}^u$ is not necessarily determined, but $\mathcal{G}$ is determined (cf. Corollary 1). Thus, it suffices to show that, for every natural number $N \in \mathbb{N}$, Maximizer has a strategy to ensure a payoff $> N$ against g in $\mathcal{G}$. Recall that strategies of Minimizer in $\mathcal{G}$ correspond to strategies in $\mathcal{G}^u$. Let thus g and N be given. Maximizer can play as follows: play as in $\alpha(g)$ until the first Call occurs. Skip the Call-Return sequence. If $\alpha(g)$ is won via the parity condition, do this infinitely often. Otherwise, wait until the winning counter d has reached a value $> N$ and a Call occurs. Take the Call and realize the payoff as required. □

We can now prove Theorem 2. Indeed, let Σ be the set of strategies winning for Minimizer in $\mathcal{G}^u$. By Equation 1 it is recognized by an automaton of the claimed size (with $m = |\mathcal{M}|$). Then, by Proposition 11, the first item of Theorem 2, holds, and by Proposition 10 the second item is true.

6 Outlook

The presented algorithm allows to solve a general class of counter parity games, which capture several cost models for dynamic systems. These cost models are also used in many online algorithms, thus it may be beneficial to apply affine counter parity games in this domain, in the spirit of [1]. A crucial part in our proof is played by games with imperfect information and imperfect recall – a class which has been studied in classical game theory, but so far received little attention in computer science. This motivates a future more systematic study of ω-regular games with imperfect recall.

References

1. Aminof, B., Kupferman, O., Lampert, R.: Formal Analysis of Online Algorithms. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 213–227. Springer, Heidelberg (2011)
2. Berwanger, D., Kaiser, Ł., Leßenich, S.: Imperfect recall and counter games. Research Report LSV-11-20, LSV, ENS Cachan, France (2011), http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2011-20.pdf
3. Brázdil, T., Forejt, V., Krcál, J., Kretínský, J., Kucera, A.: Continuous-time stochastic games with time-bounded reachability. In: Proc. of FSTTCS 2009, pp. 61–72 (2009)
4. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Generalized mean-payoff and energy games. In: Proc. of FSTTCS 2010, pp. 505–516 (2010)
5. Chatterjee, K., Henzinger, T.A., Horn, F.: The Complexity of Request-Response Games. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 227–237. Springer, Heidelberg (2011)
6. de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: Model checking discounted temporal properties. Theoretical Computer Science 345(1), 139–170 (2005)
7. Fischer, D., Grädel, E., Kaiser, Ł.: Model checking games for the quantitative μ-calculus. Theory Comput. Syst. 47(3), 696–719 (2010)
8. Fischer, D., Kaiser, Ł.: Model Checking the Quantitative μ-Calculus on Linear Hybrid Systems. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 404–415. Springer, Heidelberg (2011)
9. Horn, F., Thomas, W., Wallmeier, N.: Optimal Strategy Synthesis in Request-Response Games. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 361–373. Springer, Heidelberg (2008)
10. Kaiser, Ł., Leßenich, S.: Counting μ-calculus on structured transition systems. In: Proc. of CSL 2012. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (to appear, 2012)
11. Kupferman, O., Piterman, N., Vardi, M.Y.: From liveness to promptness. Formal Methods in System Design 34(2), 83–103 (2009)
12. Thomas, W.: Infinite Games and Verification (Extended Abstract of a Tutorial). In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 58–64. Springer, Heidelberg (2002)
13. Wan, H.: Upper bounds for Ramsey numbers R(3, 3,. . ., 3) and Schur numbers. Journal of Graph Theory 26(3), 119–122 (1997)
14. Zimmermann, M.: Optimal bounds in parametric LTL games. In: Proc. of GandALF 2011, pp. 146–161 (2011)

Drawing Planar Graphs on Points Inside a Polygon

Therese Biedl[1,*] and Peter Floderus[2,**]

[1] David R. Cheriton School of Computer Science, University of Waterloo, Canada
biedl@uwaterloo.ca

[2] Department of Mathematics, University of Lund, Sweden
Peter.Floderus@math.lu.se

Abstract. In this paper, we study the problem of drawing a given planar graph such that vertices are at pre-specified points and the entire drawing is inside a given polygon. We give a method that shows that for an n-vertex graph and a k-sided polygon, $\Theta(kn^2)$ bends are always sufficient. We also give an example of a graph where $\Theta(kn^2)$ bends are necessary for such a drawing.

Keywords: Graph drawing, Specified point set, Bounding polygon.

1 Introduction

Drawing, or *embedding*, a planar graph in the plane is a well studied problem, and proofs that we can draw planar graphs with straight lines have been around longer than computers [12,5]. One of the first algorithmic results was that all planar graphs have such a drawing using polynomially bounded integer coordinates [6,11]. Many other graph drawing results and models of straight-line drawings have been developed since.

If we introduce the possibility to draw the edges as sequences of straight line segments, i.e., as poly-lines with *bends*, then we may be able to incorporate other constraints on the drawing. The typical measure of quality is then the number of bends used while satisfying such constraints. For example, in some applications such as geographic visualization, vertices (which may represent cities) should be placed at or near a given location. Hence the following *point-embedding problem* is of interest: Given a planar graph G, a set of points S and an injective mapping $V \to S$, can we draw G without crossing such that the vertices are at the specified locations?

It is quite clear that not all graphs have such drawings with straight lines, and it can always be done if we allow sufficiently many bends. Pach and Wenger [10] studied this point-embedding problem and gave bounds on how many bends may be needed. They showed that it can always be done with $O(n^2)$ bends, and $\Omega(n^2)$ bends are required, even for a matching, for some mappings of vertices

[*] Supported by NSERC and by the Ross and Muriel Cheriton Fellowship.

[**] Research done while visiting the University of Waterloo.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 172–183, 2012.

to points. Since then, some variants of the point-embedding problem have been studied, for example when the mapping of the vertices to points can be arbitrary [8], or when it is only partially restricted [2,4].

Here we explore the point-embedding problem with the additional restriction that the drawing must be within a given polygon. Thus, we are given a planar graph G, a point set S, a polygon P that contains all points of S in its interior, and an injective mapping from the vertices to S. We would like to create a planar graph drawing of G that lies entirely inside P and that has the vertices at the specified locations. Since we require the points of S to be in the strict interior of P, it is clear that such a drawing always exists if we allow bends, but how many bends are needed?

We came across this problem when studying maps, and give two motivating examples. Consider Figure 1(left), which shows a hand-drawn (1910) cartogram of the United States, with states skewed so that the area reflects population. The adjacencies in this map are hilariously wrong (especially for Pennsylvania), but nevertheless, the reader has no difficulty in recognizing the United States, simply because the boundary is correct. We are currently doing research on how to create cartograms that use given boundaries and other identifying features such as rivers, and encounted the problem studied in the current paper as a sub-problem. A second example is in Figure 1(right), which shows a flight map of some intra-Canadian flights. Does the flight from Toronto to Sault Ste. Marie enter US airspace? How about the one from North Bay to Thunder Bay?[1] This flight map is drawn with straight lines, while flights paths are often zig-zag lines depending on location of control towers. So this map cannot be trusted to answer the question, and a map that distinguish clearly between flights that remain entirely inside Canada and those that do not may be useful.

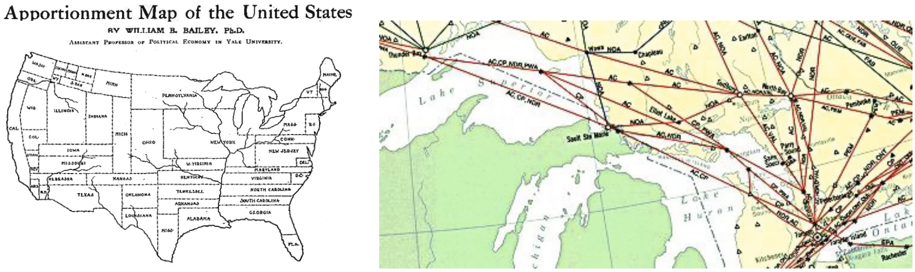

Apportionment Map of the United States
BY WILLIAM B. BAILEY, Ph.D.
Assistant Professor of Political Economy in Yale University.

Fig. 1. (left) A cartogram of the United States. (right) A flight map of parts of Canada.

The topic of drawing at fixed locations inside a polygon is also related to the *local routing* problem in VLSI design (see for example [9].) Here the modules of a chip have been placed already, and the routes of connections between the

[1] If yes, then due to the recently passed Bill C-42, data about the passengers may be forwarded to the US government, and passengers on the US no-fly list may be denied boarding. So the question may be of interest to some people.

pins of the modules must be placed in the remaining free space. However, the research on local routing is quite different from our research for two reasons: (1) Pins are located on the modules, and hence at the boundary of the drawing region. This makes planar drawings impossible in almost all cases. In contrast, we demand that points are strictly inside P, precisely so that a planar drawing is always feasible. (2) The VLSI community focuses on minimizing area as main objective. In contrast, we focused on minimizing the number of bends. (Area considerations of the resulting drawings are an interesting problem, but this remains for future studies.)

We give in this paper upper and lower bounds on the number of bends needed for a planar drawing on a given set of points inside a polygon. Our results resemble the ones by Pach and Wenger [10], but (as is to be expected) also include the size of the bounding polygon. Presume we are given an n-vertex planar graph G and a k-sided polygon P with points S inside P, and a mapping from V to S. A drawing with $O(nk)$ bends per edge would be quite easy to achieve: Take the drawing with $O(n)$ bends per edge from [10] that ignores P, and then re-route each segment of an edge to be a polyline inside P with at most k bends. We show here that we can use far fewer bends than that: there always exists a planar drawing of G inside P with vertices at pre-specified points of S that has $O(k+n)$ bends per edge. We also show that this is tight: $\Omega(n)$ edges need $\Omega(k+n)$ bends for some choice of G, S and P.

Our lower bound builds directly on the lower bound of Pach and Wenger, endowed with a suitable polygon; see Section 2. Our upper bound is also somewhat similar to the upper-bound method used in [10] where all edges are routed in parallel channels around the drawing. However, instead of "channels" we use cyclic levels, similarly as was done by Bachmeier et al. [1]. We then map the cyclic levels to line segments inside the polygon that contain the points of S and satisfy some other conditions to make such a mapping possible.

To minimize the number of bends, we need to minimize the number of times an edge needs to do a "turn", i.e., a change of direction from clockwise to counterclockwise or vice versa in a cyclic level drawing. We show that any edge needs at most two turns. This is also of interest for so-called *upward drawings* (where directed graphs are drawn such that edges go monotonically from smaller to a larger y-coordinate). Not every planar directed acyclic digraph (dag) has an upward drawing with straight lines, and testing whether it does is NP-hard [7]. Our results imply that any planar dag has a planar drawing where the y-coordinate of each tail is smaller than the y-coordinate of the head, and every edge consists of at most 3 y-monotone pieces.

2 Lower Bound

In this section, we argue our lower bounds: for some n-vertex graph graph G, point set S inside a k-sided polygon P and mapping $V \to S$, $\Omega(n)$ edges have $\Omega(n+k)$ bends each.

Our lower bound builds directly on the lower bound given by Pach and Wenger for the point-embedding problem [10]. They showed that for any planar graph that has a matching of size m, and any random assignment of the vertices to points in convex position, almost surely there are at least $\frac{m}{20}$ matching-edges that have at that have at least $m/(40)^2$ bends each.

For our lower bound, let G be the graph that is a matching of size $m := n/2$. As bounding polygon P, we use a k-sided polygon that has two regions R_1 and R_2 with $\Omega(k)$ *link-distance*, i.e., any path from a point in R_1 to a point in R_2 that stays inside P has $k/2 - 1$ bends. See Figure 2. We choose the points in S such that there are $n/2$ points in each of these two regions, and the points in S are in convex position.

Now choose a random assignment from the vertices of G to the points of S. By Pach and Wenger's result, almost surely at least $\frac{m}{20}$ edges have at least $m/(40)^2$ bends. Let E' be those edges, and let $E'' \subseteq E'$ be all those edges in E' that connect a point in R_1 with R_2. By construction, any edge in E'' has $\Omega(k)$ bends because of the link-distance, and $\Omega(m) = \Omega(n)$ bends because it is in E', and hence $\Omega(n+k)$ bends total. So all that remains to do is to bound the size of E''.

We know $|E'| \geq \frac{m}{20}$ almost surely. Any edge connects the two different regions with probability $\approx 1/2$. The expected number of edges in E'' is hence $\geq \frac{1}{2}\frac{m}{20}$. Since the variance of whether an edge connects two difference regions is $\approx \frac{1}{4}$, so the variance of $|E''|$ is $\approx \frac{1}{4}\frac{m}{2}$. Hence by Chebyshev's inequality the probability that $|E''| < \frac{1}{2}\frac{m}{20} - \sqrt{m/80}\sqrt{\frac{1}{4}\frac{m}{20}}$ is at most $80/m$. So the probability that $|E''| < m/80 = n/160$ goes to 0 as n goes to infinity. In conclusion, almost surely at least $n/160$ edges have at least $n/40 + (k-6)/2$ bends each, which proves:

Theorem 1. *Let $G = (V, E)$ be a plane graph that contains a perfect matching and P be the bounding polygon with points in it as in Figure 2. Then any random mapping of vertices to points requires $\Omega(n)$ edges to bend $\Omega(n+k)$ times almost surely.*

3 Upper Bound

To obtain an upper bound of $O(n+k)$ bends per edge, we need some intermediate results. We first give an overview of the algorithm here, and then explain the individual steps below.

So presume we are given a graph G, a point set S, a bounding polygon P, and an injective mapping of V to S. We first create an ordered set of disjoint line segments L_S (which we call *skewed levels*) inside P that contain all points of S (with at most one point per segment) and have some other useful properties. The injective mapping $V \to S$ then naturally gives an injective mapping $V \to L_S$.

Next, we consider what we call *cyclic levels*, which is a set L_C of disjoint line segments on rays from the origin. We have $|L_C| = |L_S|$, and a natural 1-to-1 correspondance $L_S \leftrightarrow L_C$, which gives an injective mapping $V \to L_C$. We want a *cyclic level drawing* of G (previously studied in [1]) where each vertex is placed somewhere on the cyclic level that it maps to. Our objective is to minimize the number of turns done by each edge. We show that every graph has a cyclic level drawing such that every edge has at most two turns.

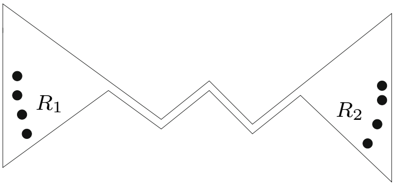


Fig. 2. The bounding polygon used in Theorem 1; it has $k/2 - 1$ reflex vertices, and any path from R_1 to R_2 must detour around all of them

With this drawing in hand, we then easily obtain a drawing of G on S inside P by mapping the cyclic levels back to the skewed levels. Since every edge has at most two turns, it crosses every level at most 3 times, and hence has $O(|L_S|) = O(k + n)$ bends.

3.1 Cyclic Level Drawings and Turns

We first start by explaining the cyclic level drawings. A *cyclic level set* $L_C = \{l_1, \ldots, l_M\}$ is a set of disjoint line segments such that l_i lies on the ray from the origin with angle $2i\pi/M$. See Figure 3. Let b_i be the endpoint of l_i closer to the origin, and t_i be the other endpoint. Note that $\{b_1, \ldots, b_M\}$ and $\{t_1, \ldots, t_M\}$ form a polygon (with one hole), which has a natural quadrangulation defined by the segments. For ease of description, we will assume that all cyclic levels have the same length and distance from the origin, so the quadrangles formed by them are trapezoids.

In the following section, we are given a set of cyclic levels, a planar graph G, and an injective mapping of vertices to cyclic levels. We want a poly-line drawing of G such that each vertex is placed on its cyclic level (but it does not matter where along that level.) Furthermore, the drawing of G should be entirely within the polygon defined by the endpoints of the levels. Cyclic level drawings of planar graphs were first studied by Bachmeier et al. [1] with the objective of testing whether a planar graph has such a drawing such that directed edges make no turns at all. In our work, we create such drawings where every edge makes at most 2 turns. Here, a *turn* is a change of direction of the edge with respect to the cyclic levels. More precisely, we say that an edge makes a turn whenever it crosses the same cyclic level twice without crossing any other cyclic level inbetween.

Theorem 2. *Given a planar graph $G = (V, E)$, a set of cyclic levels L_C and an injective mapping $l : V \to L_C$. Then G can be drawn inside the polygon formed by the cyclic levels such that any vertex v is drawn somewhere on $l(v)$, and any edge has $O(|L_C|)$ bends and at most 2 turns.*

The proof of this theorem proceeds in two steps. We first prove a weaker result, where we assume that G has a Hamiltonian cycle $C = \{v_1, v_2 \ldots v_n\}$. In this case we use a technique similar to the one used by Kaufmann and Wiese [8], who studied how to draw a planar graph on a point set but without specifying which vertex goes to which point.

Our drawing is illustrated in Figure 3. To draw C, we imagine the interior of each level l_i to be subdivided with with n points, say $p_1^{l_i}, \ldots, p_n^{l_i}$ at equal distance, in order towards the origin. Draw v_1 on the outermost point $p_1^{l(v_1)}$ of its level, v_2 on the second point $p_2^{l(v_2)}$ on its level, and so on. To route the edge (v_1, v_2), assume first that $l(v_1) > l(v_2)$. Then route the edge by adding a bend on every level i with $l(v_1) > i > l(v_2)$, placing the bend somewhere between p_1^i and p_2^i on level i. Thus we form a counter-clockwise arc from v_1. If $l(v_2) > l(v_1)$ then we route similarly but in clockwise direction from v_1, adding bends on every level i with $l(v_2) > i > l(v_1)$. All other edges of the Hamiltonian cycle are routed similar: go either counter-clockwise or clockwise along the level, in such a way that none of the added edge-segments pass through the trapezoid between the last and first level.

Finally, to close up the Hamiltonian cycle, we need to route edge (v_1, v_n); we do so by routing counter-clockwise from $l(v_1)$ (outside all points) and clockwise from $l(v_n)$ (inside all points) until both parts reach the trapezoid between the last and first level. There the parts can be connected with a segment, without introducing a crossing, since this trapezoid is empty.

Now we need to draw all remaining edges, each of which is inside or outside of the Hamiltonian cycle. Let (v_i, v_j) be an inside edge, $i < j$. Route it by going clockwise from v_i, always staying near the ith point on each level, until we are past the smallest level ℓ used by $v_{i+1}, \ldots, v_{j-1}$. Similarly route clockwise from v_j. Finally add one bend at a point just beyond ℓ and connect the two routes; edge (v_i, v_j) makes a turn at this bend. Route the outside edges similar, except go counterclockwise. All edges can be routed in this fashion with at most one turn per edge. So we have:

Lemma 1. *A planar Hamiltonian graph has a cyclic level drawing where edges of the Hamiltonian cycle have no turn and all other edges have at most one turn.*

Now we turn to graphs that are not Hamiltonian. It is well known that a planar triangulated graph is Hamiltonian if it does not contain a *separating triangle*, i.e., a triangle that has vertices both inside and outside. Moreover, a Hamiltonian cycle can be found in linear time [3]. Both Pach and Wenger [10] and Kaufmann and Wiese [8] describe methods to augment G to make it Hamiltonian in linear time. We use the method in [8], which removes a separating triangle by subdividing an edge in it, and then connects the subdivision vertex to the third vertex on the two adjacent faces. Doing this to all separating triangles results in a Hamiltonian graph that has $O(n)$ vertices and every edge has been subdivided at most once.

Thus presume we have made graph G into a Hamiltonian graph G'. Add extra cyclic levels and assign the subdivision vertices to them in an arbitrary manner. Draw G' on these cyclic levels as explained above. Then remove the added edges and restore the original edges by replacing subdivision vertices by bends. All desired properties of the resulting drawing of G are easily verified, except for

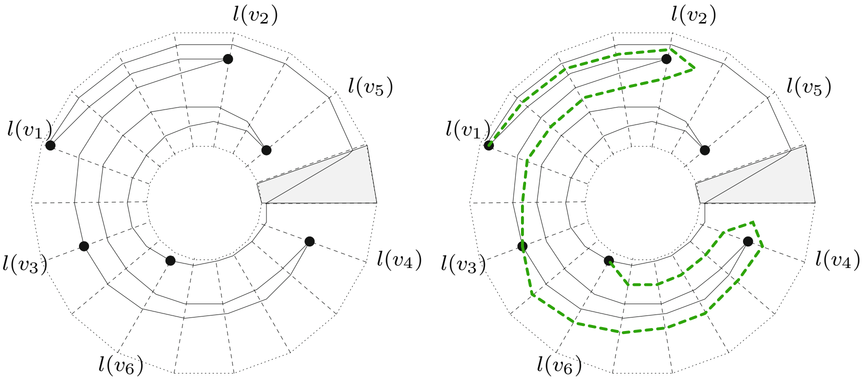


Fig. 3. (Left) The drawing of the Hamiltonian cycle. Only (v_n, v_1) is allowed to use the trapezoid between the last and first level. (Right) Adding inner and outer edges (thick dashed, green) with one turn.

the number of turns. A naive argument would say that an edge e of G has at most 3 turns, since e may have consisted of two edges e_1 and e_2 in G', each of those could have acquired one turn in G', and removing the subdivision vertex v' that was common to e_1 and e_2 might add another turn. However, in fact e has at most two turns:

- If e_1 belonged to C, then it had no turn in G', hence we have at most 2 turns total in e. Similarly we have at most 2 turns if e_2 belonged to C.
- Not both e_1 and e_2 can be outside edges. For the subdivision vertex v' has degree 4 and e_1 and e_2 are not consecutive at v', so if both e_1 and e_2 are outside edges, then there is only one edge at v' that could be on or inside the Hamiltonian cycle, but there must be at least two (the two on the Hamiltonian cycle.)

  Similarly not both e_1 and e_2 can be inside edges.
- So the only remaining case is if e_1 is an inside edge and e_2 an outside edge, or vice versa. Since we route inside edges clockwise and outside edges counter-clockwise, then e does not acquire a turn when removing the subdivision vertex v'.

We have thus proved Theorem 2: every planar graph has a drawing on cyclic levels such that every edge turns at most twice.

3.2 Skewed Levels Inside *P*

We now show how to create skewed levels inside P that contain the given points S on distinct level. The ultimate goal is to be able to map the cyclic level drawing obtained above onto these levels, which will require the following conditions:

Definition 1. *Let P be a polygon. A set L_S of line segments is called a* skewed level set *inside P if there exists polygons $P'' \subset P' \subseteq P$ and a triangulation $\mathcal{T}$ of $P' - P''$ such that:*

1. *the dual of triangulation $\mathcal{T}$ is a cycle,*
2. *every segment in L_S lies on an interior edge of $\mathcal{T}$,*
3. *for every interior edge of $\mathcal{T}$, there is exactly one edge of L_S that lies on it.*

Put differently, the segments in L_S are all disjoint, and connecting their endpoints up in order gives a polygon with exactly one hole and for which the segments define a convex quadrangulation. See also Figure 4.

Note that the dual of the triangulation defines a cylic order of the skewed levels. They could hence be viewed as a set of cyclic levels, except that they have been deformed as to fit inside P; we will exploit this corresponence later.

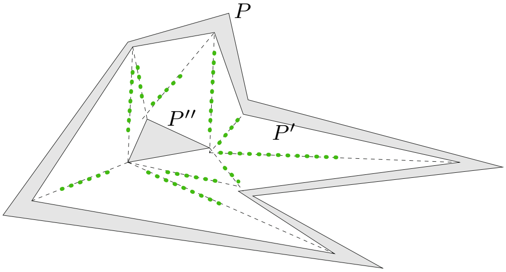


Fig. 4. A polygon P with skewed levels (thick dotted, green) that are on the inner edges of a triangulation (dashed) of the polygon $P' - P''$

Lemma 2. *Let P be a polygon and S a set of points in the interior of P. Then there exists a skewed level set of size $O(|P| + |S|)$ inside P such that each point in S is on one of the levels, and no level contains two points of S.*

Proof. To prove this, it will be helpful to assume that no three points of $S \cup P$ lie on a line, unless they all belong to S. This is not a restriction: We can change P by moving points of P inward. Recall that S is strictly inside P, so with a sufficiently small movement we still have all of S inside the new polygon, and a skewed level set inside the new polygon is also one inside P. We will also assume that P is simple (has no holes); if there is a hole then we can remove it by removing a thin channel from P that connects from the outside to the hole and does not contain a point in S. We can do this for all holes without affecting the asymptotic size of the bounding polygon.

Now triangulate P; none of the triangulation edges contains a point of S by the above. On each triangulation edge, place another *subdivision point.* Inside each face of the triangulation place a *face-point* and connect it to all polygon-corners and subdivision points of the face. We choose the positions of face-points in such a way that none of the lines incident to a face-point contains a point in S, and such that no line through two points of S contains a face-point.

The connections from face-points to the subdivision points form a tree T (which is a subdivision of the dual tree of the triangulation.) Observe that

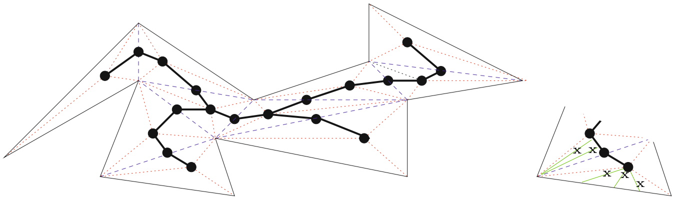


Fig. 5. A polygon (solid, black) with its triangulation (dashed, blue), the subdivision and face points (black dots), the tree between them (bold, black), and edges from the face points to the corners (dotted, red). (Right) A close-up shows how to add one skewed segment per point in S (marked by an x.)

$P - T$ is a polygon with a (degenerate) hole, and it has $O(k)$ corners. Also, the triangulation edges of P and the edges from the face points form a triangulation of the polygon $P - T$, and it is easy to see that the dual of this triangulation is a cycle (effectively, we are "walking around tree T".) See Figure 5.

For each point in $s \in S$, we now construct one line segment $l(s)$. By construction s is inside $P - T$, so it belongs to one of the triangles F in the constructed triangulation of $P - T$. One side of F belongs to either P or T; let c be the corner of F that is *not* on that edge. Let $l(s)$ be the maximal open line segment that goes through s and c and stays within F.

Point c is either a corner of P or a face point. By our assumptions on P and construction of face points, the line segment $l(s)$ therefore contains only one point from S. Our set of skewed levels now consists of these $O(n)$ line segments $l(s)$ for $s \in S$, as well as the $O(k)$ line segments in the triangulation of $P - T$ that belong to neither P nor T. One easily verifies that this is a skewed level set.[2]

3.3 Mapping the Levels

We are now ready for the main result, where we show that we can create a mapping from a drawing on cyclic levels onto the skewed levels obtained when triangulating the polygon.

Theorem 3. *Any planar graph G can be drawn with vertices at prespecified points S inside a given polygon P such that any edge has at most $O(|V| + |P|)$ bends.*

Proof. Start by creating a skewed level set L_S inside P for the point set S according to Lemma 2. Next create a set of cyclic levels L_C of cardinality $|L_S|$. Map the (cyclically ordered) set L_S to L_C, and with it, obtain a mapping from V to L_C. Create a drawing of G on L_C that has at most two turns per edge and respects this mapping (Theorem 2.)

[2] This uses a degenerate polygon for P'', which is allowed. One could make it non-degenerate by thickening T into T' and then re-triangulating $P - T'$.

Now we explain how to map this drawing back onto the skewed levels. Assume vertex v is drawn on cyclic level l_c at the point that is a λ-proportion away from the inner endpoint (i.e., if $l_c = [b, t]$ with b closer to the origin, then v is drawn at $\lambda b + (1 - \lambda)t$). Let l_s be the level that corresponds to l_c, i.e., l_s contains the point s on which we are supposed to draw v. Retract the ends of l_s in such a way that point s is a λ-proportion away from the endpoint closer to the "inner polygon" P''. Call the resulting line segment l'_s. See Figure 6.

Let l_c^1 and l_c^2 be two consecutive cyclic levels; the area R_c between them is a trapezoid. Let l_s^1 and l_s^2 be the corresponding skewed levels, and $(l_s^1)'$ and $(l_s^2)'$ be their retractions as explained above (we only retract those levels that contain a vertex.) The area R_s between $(l_s^1)'$ and $(l_s^2)'$ is a convex quadrilateral since l_s^1 and l_s^2 were on sides of a triangle by definition of a skewed level set. We now map from R_c to R_s in the obvious way, by mapping the four corners of the trapezoid to the corresponding corners in the convex quadrilateral, and mapping all other points by interpolation. See Figure 6.

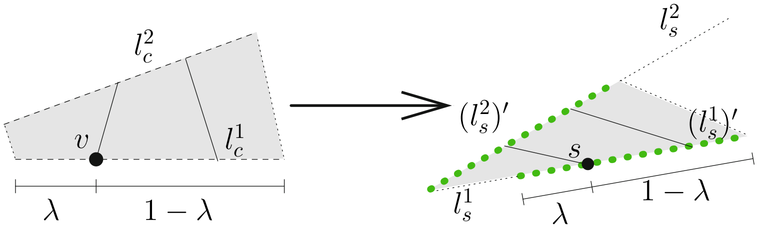


Fig. 6. Mapping the cyclic levels to skewed levels that have been retracted such that each vertex v lands on its corresponding point s

This maps each vertex to its desired point in S by construction. Let p be any point on l_c^1 where an edge e crossed l_c^1. This point maps to some point p' on $(l_s^1)'$, which will (usually) become a bend in the resulting drawing of e. Any bend that e had inside R_c (e.g., because e made a turn inside R_c) will be mapped into a bend in R_s as well. We complete the drawings of edges by putting in straight-line segments between these created bends and endpoints.

Since the drawing inside R_s is a linear mapping of the drawing in R_c, we do not create any crossing inside R_s. Since each quadrilateral of consecutive skewed levels was inside a face of a triangulation of $P' - P''$, no crossings can occur between two segments in two different quadrilaterals. Therefore we obtain a planar drawing.

In the cyclic level drawing, edges had $O(|L_C|) = O(n + k)$ bends. Mapping to the skewed levels does not introduce new bends (all edges already had bends where they crossed levels.) So every edge had $O(n + k)$ bends, which proves the theorem.

3.4 Counting the Bends

In the previous section, we were only concerned with asymptotic bounds on the number of bends. However, it is possible to give more precise bounds for the actual construction (if we do not consider the additional corners/vertices resulting from making polygon P simple or making graph G Hamiltonian.) The main observation is that bends only happen at turns or when an edge crosses a level, and we can bound these events.

Lemma 3. *For any set S of n points inside a simple k-sided polygon P, there exists a set of $n + 5k - 12$ skewed levels.*

Proof. Every point in S gives rise to one skewed level. In addition, we added skewed levels from the triangulation of P, and we count their number now.

The triangulation of P had $k - 3$ edges. Each of those obtains a subdivision point and hence gives rise to $2(k - 3)$ skewed levels. The triangulation also had $k - 2$ interior faces. Each of those obtains a face point which is connected to three corners, hence adding $3(k - 2)$ skewed levels. In total we hence have $2(k - 3) + 3(k - 2) = 5k - 12$ skewed levels.

Since every edge makes at most 2 turns, it can traverse each level at most 3 times. For each traversal, the edge may only bend $n + 5k - 12$ times, for a total of at most $3n + 15k - 36$ bends plus one bend at each turn, which gives the precise bounds for Hamiltonian graphs.

Theorem 4. *Any Hamiltonian planar graph G can be drawn with vertices at prespecified points S inside a given simple polygon P such that any edge has at most $3|V| + 15|P| - 34$ bends.*

In this paper, we studied the problem of drawing a planar graph with vertices at specified locations inside a given polygon. We provide lower bounds for the number of bends, and give a construction that matches these lower bounds asymptotically.

Our construction was done with the theoretical objective of matching the lower bounds; we make no claims as to it being esthetically pleasing or useful in practice. In particular our method of dealing with holes in the polygon by simply forbidding some region to be used at all would be unsatisfactory in a practical setting. One should also apply post-processing heuristics to remove many unnecessary bends.

We leave some open questions:

- Our lower bound uses a polygon that is unlikely to occur in practice. Can better bounds be shown for special polygons? For example, if a polygon can be split into K convex pieces (not necessarily triangles), can we create planar drawings at specified points inside the polygon with $O(n + K)$ bends per edge?
- What are area considerations? In particular, what area is required for our drawings, presuming all corners of P and points in S are at integer coordinates (say of size $O(n)$)?

References

1. Bachmaier, C., Brunner, W., König, C.: Cyclic Level Planarity Testing and Embedding. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 50–61. Springer, Heidelberg (2008)
2. Badent, M., Di Giacomo, E., Liotta, G.: Drawing colored graphs on colored points. Theoretical Computer Science 408(2-3), 129–142 (2008)
3. Chiba, N., Nishizeki, T.: The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. Journal of Algorithms 10(2), 187–211 (1989)
4. Di Giacomo, E., Didimo, W., Liotta, G., Meijer, H., Wismath, S.: Constrained Point-Set Embeddability of Planar Graphs. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 360–371. Springer, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-00219-9_35
5. Fary, I.: On straight line representation of planar graphs. Szeged. Sect. Sci. Math. 11, 229–233 (1948)
6. Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10, 41–51 (1990)
7. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. SIAM J. Comput. 31(2), 601–625 (2001)
8. Kaufmann, M., Wiese, R.: Embedding Vertices at Points: Few Bends Suffice for Planar Graphs. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 165–174. Springer, Heidelberg (1999)
9. Lengauer, T.: Combinatorial Algorithms for Integrated Circuit Layout. Teubner/Wiley & Sons, Stuttgart/Chicester (1990)
10. Pach, J., Wenger, R.: Embedding Planar Graphs at Fixed Vertex Locations. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 263–274. Springer, Heidelberg (1999)
11. Schnyder, W.: Embedding planar graphs on the grid. In: ACM-SIAM Symposium on Discrete Algorithms (SODA 1990), pp. 138–148 (1990)
12. Wagner, K.: Bemerkungen zum Vierfarben Problem. Jahresbericht Deutsch. Math. 46, 26–32 (1936)

New Advances in Reoptimizing the Minimum Steiner Tree Problem

Davide Bilò[1] and Anna Zych[2]

[1] Dip.to di Teorie e Ricerche dei Sistemi Culturali, University of Sassari, Italy
[2] Uniwersytet Warszawski
davide.bilo@uniss.it

Abstract. In this paper we improve the results in the literature concerning the problem of computing the minimum Steiner tree given the minimum Steiner tree for a similar problem instance. Using a σ-approximation algorithm computing the minimum Steiner tree from scratch, we provide a $\left(\frac{3\sigma-1}{2\sigma-1}+\epsilon\right)$ and a $\left(\frac{2\sigma-1}{\sigma}+\epsilon\right)$ -approximation algorithm for altering the instance by removing a vertex from the terminal set and by increasing the cost of an edge, respectively. If we use the best up to date $\sigma = \ln 4+\epsilon$, our ratios equal 1.218 and 1.279 respectively.

1 Introduction

The concept of reoptimization adds an interesting twist to hard optimization problems. It is motivated by the fact, that when confronted with a hard problem in reality, one often has some additional information about an instance at hand. Imagine for example a train station where the train traffic is regulated via a certain time schedule. The schedule is computed when the station is ready to operate and, provided that an unexpected event does not occur, there is no need to compute it again. Unfortunately an unexpected event, such as a delayed train, is in a long run inevitable. Reoptimization addresses this scenario, asking whether knowing a solution for a certain problem instance is beneficial when computing a solution for a similar instance. When dealing with relatively stable environments, i. e., where changing conditions alter the environment only for a short period of time, it seems reasonable to spend even a tremendous amount of time on computing a good solution for the undisturbed instance, and profit from it when confronted with a temporal change.

The term reoptimization was mentioned for the first time in [20] and applied to the problem of scheduling with forbidden sets for the scenarios of adding or removing a forbidden set. Since then, the concept of reoptimization has been successfully applied to various problems like the Traveling Salesman problem [3,1,8,11], the Steiner Tree problem [22,5,9,13,16], the Knapsack problem [2], the Weighted Minimum Set Cover problem [17], various covering problems [7], and the Shortest Common Superstring problem [6]. A survey of reoptimization problems can be found in [10,15].

The Minimum Steiner Tree (*SMT*) problem is a very prominent optimization problem with many practical applications, especially in network design. It is

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 184–197, 2012.

APX-complete [4]. The best up to date approximation ratio is due to a randomized $\ln 4 + \epsilon$-approximation algorithm [12]. The best deterministic algorithm achieves an approximation ratio of 1.55 [19]. The problem of reoptimizing *SMT* where the modification of an instance consists of adding or removing one vertex to/from the input graph was considered in [14]. The modifications of adding or removing a vertex to/from the terminal set and increasing or decreasing the cost of an edge was addressed in [9,5]. All these reoptimization problems are strongly NP-hard [9].

We improve in this paper the best up to date results for *SMT* under removing a terminal from the terminal set and for *SMT* under increasing the cost of an edge, complementing the new results in [22] for *SMT* under adding a terminal to the terminal set. We achieve a 1.218 approximation ratio for the scenario of removing a terminal and 1.279 for the scenario of edge cost increase. For the terminal addition and edge cost decrease the best ratios remain 1.218 [22] and 1.246 [5] respectively.[1] It is worth to remark that, contrary to [5], for the modification of decreasing the cost of an edge we do not assume that the modified graph remains metric.

For both reoptimization scenarios considered in this paper we employ the technique developed in [22]. Nevertheless, for both scenarios, the technique cannot be applied directly as for the scenario of adding a terminal to the terminal set. The heart of the paper is Lemma 3, which settles an important property that allows successfully applying the technique.

2 Preliminaries

Let us begin with a formal definition of the Minimum Steiner Tree Problem (*SMT*). We call a complete graph $G = (V, E, \text{cost})$ with the cost function $\text{cost} : E \to \mathbb{Q}^+$ *metric* if the edge weights satisfy the *triangle inequality*, i.e, $\text{cost}(\{u, v\}) \leq \text{cost}(\{u, w\}) + \text{cost}(\{w, v\})$ for all $u, v, w \in V$.

Definition 1. ***Minimum Steiner Tree Problem (SMT)***

Instance: *a metric graph* $G = (V, E, \text{cost})$ *and a terminal set* $S \subseteq V$*;*
Solution: *a Steiner tree, i.e., a subtree of* G *containing* S*;*
Objective: *minimize the sum of the costs of the edges in the solution.*

We view an algorithm as a function from the instance space to the solution space. Some of our results use an approximation algorithm for *SMT* as a subroutine and refer to it as ApprSMT. Any of the approximation algorithms for *SMT* can be used as ApprSMT.

1. ***The Minimum Steiner Tree Terminal Removal problem***
   Instance: a metric graph $G = (V, E, \text{cost})$, a terminal set S, an optimal solution Opt to (G, S), and a modified terminal set $S' = S \setminus \{t\}$ for some $t \in S$;
   Solution: a Steiner tree for (G, S').

[1] In [5] the authors provide a $\frac{5\sigma-3}{3\sigma-1}$-approximation algorithm, where σ is the approximation ratio of an algorithm for the Steiner tree problem.

2. ***The minimum Steiner Tree Edge Cost Increase problem***
   Instance: a metric graph $G = (V, E, \text{cost})$, a terminal set $S \subseteq V$, an optimal solution OPT to (G, S), and a modified graph $G' = (V, E, \text{cost}')$, where cost' coincides with cost on all edges but one edge e^* and $\text{cost}(e^*) < \text{cost}'(e^*)$;
   Solution: a Steiner tree for (G', S).

We introduce the notation used in the paper. Given a simple graph G, we denote the set of its vertices by $V(G)$ and the set of its edges by $E(G)$. We denote an (undirected) edge $e \in E(G)$ by $\{u, v\}$, where $u, v \in V(G)$ are the endpoints of e. If H is a subgraph of G, we write $H \subseteq G$. An edge $\{u, v\} \in E(G)$ can be identified with a subgraph $H \subseteq G$ with $V(H) = \{u, v\}$ and $E(H) = \{\{u, v\}\}$. The neighborhood $\mathit{\Gamma}_G(v)$ of a vertex $v \in V(G)$ in a graph G is the set of vertices adjacent to v in $E(G)$. The degree of a vertex $v \in V(G)$ is defined as the size of its neighborhood: $deg_G(v) = |\mathit{\Gamma}_G(v)|$. For two subgraphs $G_1, G_2 \subseteq G$ we denote by $G_1 \cup G_2$ a graph G' such that $V(G') = V(G_1) \cup V(G_2)$ and $E(G') = E(G_1) \cup E(G_2)$. For two subgraphs $G_1, G_2 \subseteq G$ we denote by $G_1 - G_2$ a graph G' such that $E(G') = E(G_1) \setminus E(G_2)$ and $V(G')$ is $V(G_1)$ without isolated vertices. To analyze our algorithms, we often refer to costs of graphs rather than edges. The cost of graph $G_1 \subseteq G$ is denoted by $\text{cost}(G_1)$ and is the sum of the costs of all edges in $E(G_1)$.

A path $P = (V(P), E(P))$ is a connected graph in which two vertices have degree one (called the endpoints of the path) and the remaining ones have degree two. The length of a path is its number of edges. In a shortest path, the length of the path is minimized whereas in a cheapest path its cost is minimized. A forest is a graph that consists of a union of disjoint trees: $F = T_1 \cup \cdots \cup T_m$. We denote the number of trees in F as $|F|_T$.

For a graph $G = (V, E, \text{cost})$ with an arbitrary edge weight function $\text{cost} : E \to \mathbb{Q}^+$, we denote by $\overline{G} = (V, \{\{u, v\} \mid u, v \in V, u \neq v\}, \overline{\text{cost}})$ the *metric closure* of G, i.e., the complete graph on $V(G)$ where $\overline{\text{cost}}(\{u, v\})$ is defined as the cost of the cheapest path in G from u to v. Observe that $\overline{G}$ is metric. The optimal Steiner tree for an arbitrarily edge weighted graph G and for its metric closure $\overline{G}$ coincide due to Lemma 1, whose proof can be found in [18].

Lemma 1. *Let $G = (V, E, \text{cost})$ be an edge-weighted graph and $S \subseteq V$ be a set of terminals. If a tree T is a minimum Steiner tree for (G, S) then it is also a minimum Steiner tree for $(\overline{G}, S)$. Moreover, any Steiner tree T' for $(\overline{G}, S)$ can be easily transformed to a Steiner tree for (G, S) of cost less than or equal to the cost of T'.*

The lemma above implies that for both *SMT* problem and *SMT* under removing a vertex from the terminal set the restriction on the input graph to be metric and complete does not cause any loss of generality. A similar restriction for the scenario of edge cost increase is discussed in Section 4.

Observation 1. *Due to the metricity and completeness of the graph we may assume without loss of generality that $deg_{\text{OPT}}(v) \geq 3$ for an optimal solution* OPT *to any SMT problem instance (G, S) and any $v \in V(\text{OPT}) \setminus S$.*

3 The *SMT* Terminal Removal Problem

Before we move on to the *SMT* Terminal Removal problem, we introduce a few techniques and observations that will be used later. We start with the technique introduced in [21], which relies on executing an approximation algorithm for *SMT* on a reduced instance and expanding the obtained solution. The technique is based on the procedure $\mathcal{S}_{\textsc{ApprSMT}}$ presented in Algorithm 1. This procedure, for a given instance (G, S) of *SMT* and a forest F of trees contained in G, contracts the edges of F. Routine Δ, given an *SMT* instance and an edge, computes another, reduced instance of *SMT*, where the edge is contracted to a single node which becomes a terminal in the reduced instance. Additionally, if the contraction creates multiple edges, the cheapest one is chosen for each pair of vertices (in other words, the metric closure is computed). Then a σ-approximation algorithm ApprSMT for *SMT* is applied on the obtained reduced instance. After adding the contracted edges to the resulting solution, the modified solution becomes feasible for the original instance.

Algorithm 1. $\mathcal{S}_{\textsc{ApprSMT}}$

```
Input: A metric graph $G$, a terminal set $S \subseteq V(G)$, a forest $F = T_1 \cup \cdots \cup T_k$
$(G_0, S_0) := (G, S)$
for all edges $e$ in $E(F)$ do
    $(G_j, S_j) := \Delta((G_{j-1}, S_{j-1}), e_j)$
end for
$T := \textsc{ApprSMT}(\overline{G_k}, S_k)$
Output: $T \cup F$
```

Lemma 2. *If* $F \subseteq \textsc{Opt}(G, S)$ *then the algorithm* $\mathcal{S}_{\textsc{ApprSMT}}((G, S), F)$ *returns a solution of cost at most* $\sigma\text{Opt}(G, S) - (\sigma - 1)\text{cost}(F)$ *where* σ *is the approximation ratio of* ApprSMT. *Proof to be found in [5,21].*

The next remark states that it is enough to consider the case when the removed terminal t has at least two neighbors in Opt.

Remark 1. Let Opt be the optimal solution to (G, S) given in the input of the *SMT* under removing a vertex from the terminal set. Let $t \in S$ be the terminal removed from S, i.e., $S' = S \setminus \{t\}$. We may assume that $deg_{\textsc{Opt}}(t) \geq 2$.

Proof. If there is only one edge $e = \{t, v\}$ adjacent to t in Opt, then Opt $- e$ is an optimal solution to $(G, S \setminus \{t\} \cup \{v\})$. If $v \in S'$ then Opt $- e$ is optimal to (G, S'). Otherwise we build a new instance for the *SMT* under removing a vertex from the terminal set. We let the non-modified instance to be $(G, S \setminus \{t\} \cup \{v\})$ with the optimal solution Opt $- e$. Then we let the terminal to be removed be v, so the modified set of terminals is $S \setminus \{t\}$ which is equal to S'. For the instance we have it holds that $deg_{\textsc{Opt}-e}(v) \geq 2$. □

Below we introduce a lemma that settles an important property of $\textsc{Opt}'$. The lemma holds for any optimal Steiner tree, but we formulate it using $\textsc{Opt}'$, because we want to apply it to $\textsc{Opt}'$. We define a binary tree as a tree where there is just one vertex of degree 2 called root, and every other vertex has either degree 1 (a leaf) or 3 (an inner vertex). The set of leaves of a tree T is denoted by $\mathrm{Leaf}(T)$ and the set of inner vertices by $\mathrm{Inner}(T)$.

Definition 2. *Let* $\textsc{Opt}'$ *be a minimum Steiner tree for an SMT problem instance* (G, S'). *A binary subtree* $B \subseteq \textsc{Opt}'$ *of* $\textsc{Opt}'$ *is* A*-appropriate for a set* $A \subseteq S'$ *if and only if* $\mathrm{Leaf}(B) \subseteq A$ *and* $\mathrm{Inner}(B) \cap S' = \emptyset$.

In Figure 1 we illustrate Definition 2 with an example of an optimal tree $\textsc{Opt}'$ for an *SMT* instance (G, S'), containing an A-appropriate subtree B for some $A \subseteq S'$. The next lemma shows that for any instance (G, S') of the *SMT* problem and any partition of the terminal set S' into $S' = S_1 \cup S_2$ (i. e., $S_2 = S' \setminus S_1$), an optimal solution $\textsc{Opt}'$ (satisfying Observation 1) contains a subtree $P^{(3)} \cup B_1 \cup B_2$. The subtree $P^{(3)} \cup B_1 \cup B_2$ we prove to exist consists of two binary trees B_1 and B_2, that are S_1 and S_2-appropriate respectively, and a path $P^{(3)}$ of at most three edges connecting these two trees. Figure 2 shows the desired subtree.

Lemma 3. *Let* $\textsc{Opt}'$ *be a minimum Steiner tree for* (G, S') *satisfying Observation 1. Let* $S' = S_1 \cup S_2$*, where* $S_2 = S' \setminus S_1$*, be any partition of a terminal set into two disjoint subsets. Then* $\textsc{Opt}'$ *contains a subtree* $P^{(3)} \cup B_1 \cup B_2$ *such that*

- $|P^{(3)}| \leq 3$,
- B_1 *is* S_1*-appropriate tree rooted at* r_1*, where* $r_1 \in \mathrm{Leaf}(P^{(3)})$,
- B_2 *is* S_2*-appropriate tree rooted at* r_2*, where* $r_2 \neq r_1$*,* $r_2 \in \mathrm{Leaf}(P^{(3)})$.

Proof. Let $(\overline{S_1}, \overline{S_2})$ be minimum size cut in $\textsc{Opt}'$ between S_1 and S_2, i.e., a partition of vertices $V(\textsc{Opt}') = \overline{S_1} \cup \overline{S_2}$ into two disjoint subsets $\overline{S_1} \cap \overline{S_2} = \emptyset$ such that $S_1 \subseteq \overline{S_1}$ and $S_2 \subseteq \overline{S_2}$ and the number of edges of $\textsc{Opt}'$ having one endpoint in $\overline{S_1}$ and the other in $\overline{S_2}$ is minimum. Let C be a bipartite graph consisting of edges crossing the cut and their endpoints. Hence, $C \subseteq \textsc{Opt}'$ is a (not necessarily

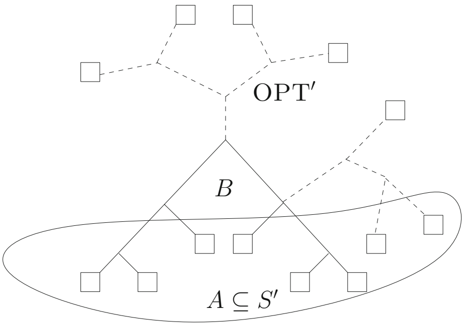


Fig. 1. An A-appropriate subtree $B \subseteq \textsc{Opt}'$, marked with solid line

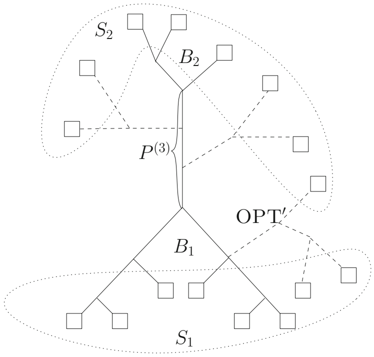


Fig. 2. A subtree $P^{(3)} \cup B_1 \cup B_2 \subseteq$ Opt′ proven to exist in Lemma 3

induced) subgraph of Opt′. Let $W_1 = V(C) \cap \overline{S_1}$ and $W_2 = V(C) \cap \overline{S_2}$ be the partition of vertices in C into two sets independent in C. Note that W_1 and W_2, although independent in C, do not have to be independent in Opt′. For each vertex $v \in W_j$, $j \in \{1, 2\}$, let $D(v)$ be the set of edges in Opt′ between v and the vertices of $\overline{S_j}$. Let $D(W_j) = \bigcup_{v \in W_j} D(v), j \in \{1, 2\}$. We illustrate the situation in Figure 3. We say that an edge $e \in D(W_j)$, $j \in \{1, 2\}$ is binary, if its endpoint not in W_j is a root of an S_j-appropriate subtree. For the sake of convenience, if $v \in W_j$ is a terminal, we let it have an imaginary binary edge in $D(v)$ incident only to v. We aim to prove that there are two binary edges, $f_1 \in D(W_1)$ and $f_2 \in D(W_2)$, both adjacent to the same edge $f_c \in C$.

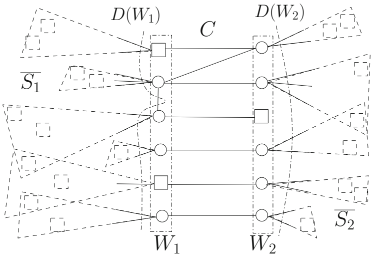


Fig. 3. Illustration to Lemma 3: Opt′ with its corresponding structures

Observe, that for each non-terminal vertex $v \in W_j$, $j \in \{1, 2\}$, it holds that $|D(v)| \geq 2$: if v is an endpoint of one edge in C, then it must have two adjacent edges in $D(v)$ as $deg_{\text{Opt}'}(v) \geq 3$; otherwise, due to the minimality of the cut C, vertex v must have at least as many adjacent edges outside the cut as inside the cut.

We now define an equivalence relation $\approx_j$, $j \in \{1, 2\}$ on the vertices of W_j. We say that $v \approx_j w$ for $v, w \in W_j$ if there is a path from v to w in Opt′ that contains only vertices in $\overline{S_j}$. We call such a path an $\overline{S_j}$-path. Hence, the vertices in W_j are partitioned into equivalence classes $[\cdot]_{\approx_j}$ with respect to relation $\approx_j$. As a first step, we prove that for each class $R \in [\cdot]_{\approx_j}$, the set of edges $D(R) = \bigcup_{v \in R} D(v)$ contains at least one binary edge.

First observe that, if R contains a terminal vertex, than we are done because of the imaginary binary edges. So assume that $R \cap S' = \emptyset$. Let us start with vertex $v_1 \in R$ and consider an edge $e_1 \in D(v_1)$. If e_1 is a binary edge, we are done. Otherwise, there is an $\overline{S_j}$-path starting with e_1 from v_1 to another vertex v_2 in R . We follow this path to v_2 and consider all the vertices visited. We enter vertex v_2 with edge $f_2 \in D(v_2)$. Now, since $|D(v_2)| \geq 2$, there is another edge $e_2 \in D(v_2)$. If e_2 is binary, we are done again. Otherwise, there is an $\overline{S_j}$-path starting with e_2 from v_2 to some $v_3 \in R$. We continue following the paths until either we found a binary edge, or we hit a vertex that we already visited. Since we always leave a vertex via a different edge as we entered, the latter implies a cycle in $\textsc{Opt}'$ and hence a contradiction.

Now let $[\cdot]_{\approx_1} = \{\zeta_1, \ldots \zeta_l\}$ and $[\cdot]_{\approx_2} = \{\xi_1 \ldots \xi_m\}$ be the equivalence classes on W_1 and W_2 respectively. Note that there can be at most one edge in C between ζ_i and ξ_j, as otherwise $\textsc{Opt}'$ contains a cycle. Recall that any vertex in W_1 is connected with an edge in C to a vertex in W_2 and vice-versa. We call a vertex binary if it is adjacent to a binary edge. We proved that every ζ_i and every ξ_j contains at least one binary vertex. Let $v \in \zeta_1$ be a binary vertex in ζ_1. It is adjacent via an edge $e \in C$ to some vertex $w \in \xi_j$. If w is binary, we let $f_c = e$ and the proof is completed. Otherwise, there is another vertex in ξ_j that is binary, call it w'. Vertex w' is connected to v' in ζ_i, $i \neq 1$. If v' is not binary, there must be another connection going from a binary vertex in ζ_i to another yet unvisited ξ_j. We can continue traversing the equivalence classes in this way until either we hit a binary vertex, or a class that was already visited. The latter is a contradiction, as it implies that there is a cycle in $\textsc{Opt}'$. □

Once we have the above lemma at our disposal, we propose the approximation algorithm for The Minimum Steiner Tree Terminal Removal problem. We start with an intuitive description of the proposed algorithm. The algorithm is parametrized with parameters Z and Y. Let $\textsc{Opt}$ be rooted at terminal t that we want to remove. Due to Remark 1, we may assume that t has at least two sons t_1 and t_2 in $\textsc{Opt}$. Each of these two sons determines a subtree of $\textsc{Opt}$, i. e. the subtree containing t_i, $i \in \{1, 2\}$, obtained by removing the edge between t and t_i from $\textsc{Opt}$. We let $S'_i \subseteq S'$, $i \in \{1, 2\}$, be the set of terminals contained in the subtree of t_i. Clearly, $S'_1 \cap S'_2 = \emptyset$. Let P_t be the cheapest path from t to S'_1 in $\textsc{Opt}$ and let R_t be the cheapest path from t to S'_2 in $\textsc{Opt}$. Let $P_t^{(Y)} \subseteq P_t$, respectively $R_t^{(Y)} \subseteq R_t$ be the path composed of the first Y edges on P_t, respectively R_t, starting from t. Let P'_t and R'_t be the remaining parts of P_t and R_t, i. e., $P_t = P_t^{(Y)} \cup P'_t$ and $R_t = R_t^{(Y)} \cup R'_t$. If P_t (respectively R_t) is shorter than Y, we let $P_t^{(Y)} = P_t$ (respectively $R_t^{(Y)} = R_t$) and P'_t (respectively R'_t) be empty. The reoptimization algorithm removes paths $P_t^{(Y)}$ and $R_t^{(Y)}$ from $\textsc{Opt}$, creating a forest F of at most $2Y + 1$ trees. Note that every tree in F contains at least one terminal vertex. The algorithm then applies the procedure Connect, described further in more detail (see Algorithm 2), to connect the obtained forest. The procedure Connect on the one hand guesses how Opt′ connects the trees in F, on the other hand it uses $\mathcal{S}_{\text{ApprSMT}}$ to self-reduce the instance using the guessed edges of Opt′. At the end Connect returns the best solution among those it computes. The reoptimization algorithm then returns a better solution among Opt and the output of Connect.

We now describe the procedure Connect in more detail. It takes as an input an *SMT* instance (G, S'), a forest $F = T_1 \cup \cdots \cup T_m$ and a parameter Z. The precondition for Connect is that each tree in F has to contain a terminal vertex from S'. Procedure Connect computes a number of Steiner trees for (G, S') and returns the best among the computed trees. It connects the trees in F in a recursive manner. At each recursive call it decreases the number of trees in F by connecting the last tree T_m, i.e., the tree with the highest index among the trees in F, to some other tree $T_i, i < m$. This is done with the help of Lemma 3. Since every tree T_i contains a terminal, we can partition the terminal set S' into two non-empty sets of terminals: the terminals in the last tree T_m and the terminals in the remaining trees of F. To be more precise, we set $S_1 = S' \cap V(T_m)$ and $S_2 = S' \cap V(F - T_m)$ to be the partition of S'. Then Opt′ contains a subtree $B_1 \cup P^{(3)} \cup B_2$ as in Lemma 3. Procedure Connect guesses $B_1^{(Z)} \cup P^{(3)} \cup B_2^{(Z)}$ by iterating through all trees of appropriately bounded size. There $B_j^{(Z)}$, $j \in \{1, 2\}$, is the subtree of the first Z floors of the tree B_j. If the height of B_j is smaller than Z, we let $B_j^{(Z)} = B_j^{(h)}$, where $h < Z$ is the height of B_j. In other words, if there is a terminal within the first Z floors of B_j, we only guess as far as to reach that terminal. The number of edges in the tree $B_1^{(Z)} \cup P^{(3)} \cup B_2^{(Z)}$ to be guessed is bounded by $|E(B_1^{(Z)} \cup P^{(3)} \cup B_2^{(Z)})| \leq 2^{Z+1} + 3$. Hence, we let $\overline{T}_m$ run trough all trees of size bounded by $2^{Z+1} + 3$. For each $\overline{T}_m$ we compute the cheapest edges e_m and f_m connecting $\overline{T}_m$ with S_1 and S_2 respectively, and use $\overline{T}_m \cup e_m \cup f_m$ to connect T_m to some other tree $T_i, i < m$. We proceed recursively with a forest of a smaller size. Once F becomes a single connected tree T_1 (at the bottom of recursion), two solutions are computed: $\text{Sol}_j = T_1$ and $\text{Sol}'_j = \mathcal{S}_{\text{ApprSMT}}((G, S'), \bigcup_{\overline{T}_i \in Q} \overline{T}_i)$. There, j is incremented whenever a new pair of solutions is computed and Q is a stack that stores the trees from higher level recursive calls. We present the pseudo-code in Algorithm 2. We allow two operations on the stack Q: $\leftarrow T$ puts a tree T on the top of the stack, whereas $\rightarrow$ pops a tree from the top of the stack.

In the following lemma we give the bounds on the costs of solutions computed by Connect.

Lemma 4. *Let (G, S') be an instance of the SMT problem where G is metric and* Opt′ *be an optimal solution to it satisfying Observation 1. Let a forest $F = T_1 \cup \cdots \cup T_m$ be such that for every tree $T_i, i \in \{1, \ldots, m\}$, we have $V(T_i) \cap S' \neq \emptyset$. Let Z be a positive integer. Let* Sol *and* Sol′ *be the best solution among* Sol_j *and* Sol'_j*, respectively. Then there exists a forest $\overline{F} \subseteq$* Opt′ *of* Opt′ *such that:*

Algorithm 2. Recursive procedure Connect

```
Input: An SMT instance $(G, S')$, $F = \bigcup_{i=1}^{m} T_i$, parameter $Z$
if $m := |F|_T > 1$ then
    $S_1 := S' \cap V(T_m)$
    $S_2 := S' \cap V(\bigcup_{i=1}^{m-1} T_i)$
    $\mathcal{G}_m :=$ all subtrees of $G$ of at most $2^{Z+1} + 3$ edges
    for each tree $\overline{T}_m \in \mathcal{G}_m$ do
        $Q \leftarrow \overline{T}_m$
        $e_m :=$ the cheapest edge between $\overline{T}_m$ and $S_1$
        $f_m :=$ the cheapest edge between $\overline{T}_m$ and $S_2$
        Let $i$ be the index of $T_i \in F$ which contains an endpoint of $f_m$
        $T_i := T_i \cup T_m \cup e_m \cup f_m \cup \overline{T}_m$
        Connect$((G, S'), F - T_m, Z)$
        $Q \rightarrow$
    end for
else
    $j := j + 1$; Sol$_j := T_1$;
    Sol$'_j := \mathcal{S}_{\text{ApprSMT}}((G, S'), \bigcup_{\overline{T}_i \in Q} \overline{T}_i)$;
end if
Output: the best among the computed solutions Sol$_j$ and Sol$'_j$
```

$$\text{cost}(\text{Sol}) \leq \text{cost}(F) + \text{cost}(\overline{F}) + |F|_T \frac{\text{cost}(\text{Opt}')}{2^Z},$$
$$\text{cost}(\text{Sol}') \leq \sigma\text{cost}(\text{Opt}') - (\sigma - 1)\text{cost}(\overline{F}).$$

Moreover, the running time of Connect *is polynomial if Z and the number of trees in F are constant.*

Proof. At each recursive call the respective $\overline{T}_i$ runs through all trees of size at most $2^{Z+1}+3$. Due to Lemma 3, for the partition $S' = S_1 \cup S_2$ computed in that recursive call, there exist a tree $B_1 \cup P^{(3)} \cup B_2 \subseteq \text{Opt}'$. Since $B_1^{(Z)} \cup P^{(3)} \cup B_2^{(Z)} \subseteq B_1 \cup P^{(3)} \cup B_2 \subseteq \text{Opt}'$ is a subtree of size at most $2^{Z+1}+3$, at some point $\overline{T}_i$ is set to $B_1^{(Z)} \cup P^{(3)} \cup B_2^{(Z)}$. Hence, at some point at the bottom of the recursion we end up with every $\overline{T}_i \in Q$ set to its respective $B_1^{(Z)} \cup P^{(3)} \cup B_2^{(Z)}$. In what follows, we analyze the solutions computed exactly at that point. Let j be such that Sol$_j$ and Sol$'_j$ were computed at that point and let $\overline{F} = \bigcup_{i=1}^{|F|_T} \overline{T}_i$ be the forest composed of the corresponding selection of trees on Q. The following bound holds:

$$\text{cost}(\text{Sol}_j) \leq \text{cost}(F) + \text{cost}(\overline{F}) + \sum_{i=1}^{|Q|}(\text{cost}(e_i) + \text{cost}(f_i)).$$

Note that if there is a terminal leaf in $B_1^{(Z)} \subseteq \overline{T}_i$ then $\text{cost}(e_i) = 0$. If there is a terminal leaf in $B_2^{(Z)} \subseteq \overline{T}_i$ then $\text{cost}(f_i) = 0$. If in $B_i^{(Z)}$, $i \in \{1, 2\}$, there is no terminal, due to Lemma 3 there are two paths branching from each of 2^Z non-terminal leaves of $B_i^{(Z)}$. So $\mathrm{cost}(e_i) \leq \frac{\mathrm{cost}(\textsc{Opt}')}{2^{Z+1}}$ (and analogously $\mathrm{cost}(f_i) \leq \frac{\mathrm{cost}(\textsc{Opt}')}{2^{Z+1}}$). In any case $\max\{\mathrm{cost}(e_i), \mathrm{cost}(f_i)\} \leq \frac{\mathrm{cost}(\textsc{Opt}')}{2^{Z+1}}$. Since $|Q| \leq |F|_T$ and $\mathrm{cost}(\textsc{Sol}) \leq \mathrm{cost}(\textsc{Sol}_j)$, the upper bound on $\mathrm{cost}(\textsc{Sol})$ as claimed in the lemma follows. The second inequality claimed in the lemma is a consequence of Lemma 2 and the fact that $\overline{F} \subseteq \textsc{Opt}'$.

Finally, let us analyze the running time of the procedure Connect. At each recursive call Connect runs through at most $|E(G)|^{(2^{Z+1}+3)}$ trees and makes a recursive call for each of these trees. The depth of the recursion is bounded by the number of trees in the input forest minus one, i. e., $|F|_T - 1$. Hence, we obtain a running time of $\mathcal{O}(|I|^{(2^{Z+1}+3)(|F|_T-1)} \cdot \mathrm{Poly}(|I|))$ for some polynomial function Poly where $|I|$ is the size of the instance. If Z and $|F|_T$ are constant, then the running time is polynomial. □

The next lemma states the main result of this section.

Lemma 5. *For any constant $\epsilon > 0$, there is a polynomial-time $(\frac{3\sigma-2}{2\sigma-1} + \epsilon)$-approximation algorithm for SMT under removing a terminal.*

Proof. The final output of the reoptimization algorithm for the modification of removing a terminal is the best solution among $\textsc{Connect}((G, S'), \textsc{Opt} - (P_t^{(Y)} \cup R_t^{(Y)}), Z)$ and $\textsc{Opt}$.

Due to Lemma 4 and because $|F|_T \leq 2Y + 1$, the bounds on the computed solutions are as follows:

$$\mathrm{cost}(\textsc{Sol}) \leq \mathrm{cost}(\textsc{Opt} - (P_t^{(Y)} \cup R_t^{(Y)})) + \mathrm{cost}(\overline{F}) + \frac{(2Y+1)\mathrm{cost}(\textsc{Opt}')}{2^Z},$$
$$\mathrm{cost}(\textsc{Sol}') \leq \sigma\mathrm{cost}(\textsc{Opt}') - (\sigma - 1)\mathrm{cost}(\overline{F}),$$
$$\mathrm{cost}(\textsc{Opt}) \leq \mathrm{cost}(\textsc{Opt}') + \min\{\mathrm{cost}(P_t), \mathrm{cost}(R_t)\}.$$

Taking into account the bound $\mathrm{cost}(\textsc{Opt}) \leq \mathrm{cost}(\textsc{Opt}') + \mathrm{cost}(R_t)$ it holds that

$$\begin{aligned} &\mathrm{cost}(\textsc{Opt} - (P_t^{(Y)} \cup R_t^{(Y)})) \\ &\leq \mathrm{cost}(\textsc{Opt}) - \mathrm{cost}(P_t^{(Y)}) - \mathrm{cost}(R_t^{(Y)}) \\ &\leq \mathrm{cost}(\textsc{Opt}') + \mathrm{cost}(R_t) - \mathrm{cost}(R_t^{(Y)}) + \mathrm{cost}(P_t) - \mathrm{cost}(P_t^{(Y)}) - \mathrm{cost}(P_t) \\ &= \mathrm{cost}(\textsc{Opt}') + \mathrm{cost}(R_t') + \mathrm{cost}(P_t') - \mathrm{cost}(P_t). \end{aligned}$$

Since w.l.o.g. $\mathrm{cost}(R_t') \leq \mathrm{cost}(P_t')$, we can rewrite

$$\mathrm{cost}(\textsc{Sol}) \leq (1 + \frac{2Y+1}{2^Z})\mathrm{cost}(\textsc{Opt}') - \mathrm{cost}(P_t) + 2\mathrm{cost}(P_t') + \mathrm{cost}(\overline{F}),$$

where $\mathrm{cost}(P_t') \leq \frac{\mathrm{cost}(\textsc{Opt})}{Y} \leq \frac{\mathrm{cost}(\textsc{Opt}')}{Y} + \frac{\mathrm{cost}(P_t)}{Y}$ (due to Observation 1 and the fact that P_t was the cheapest, there have to be $Y - 1$ paths in Opt more expensive than P_t' and disjoint with it) and the bound on the cost of Opt given by $\mathrm{cost}(\textsc{Opt}) \leq \mathrm{cost}(\textsc{Opt}') + \mathrm{cost}(P_t)$.

As a result of these calculations, we obtain the following bounds on the considered solutions:

$$\begin{aligned}
\mathrm{cost}(\textsc{Sol}) &\le (1+\frac{2}{Y}+\frac{2Y+1}{2^Z})\mathrm{cost}(\textsc{Opt}')-(1-\frac{2}{Y})\mathrm{cost}(P_t)+\mathrm{cost}(\overline{F}),\\
\mathrm{cost}(\textsc{Sol}') &\le \sigma\mathrm{cost}(\textsc{Opt}')-(\sigma-1)\mathrm{cost}(\overline{F}),\\
\mathrm{cost}(\textsc{Opt}) &\le \mathrm{cost}(\textsc{Opt}')+\mathrm{cost}(P_t).
\end{aligned}$$

Hence the cost of the overall solution is bounded by minimum among these three bounds. Observe that the first two bounds are functions of $\mathrm{cost}(\overline{F})$, increasing and decreasing with $\mathrm{cost}(\overline{F})$ respectively. Setting them equal allows computing the maximum value of the minimum among the first two bounds and the value of $\mathrm{cost}(\overline{F})$ where the maximum is reached as a function of $\mathrm{cost}(\textsc{Opt}')$ and $\mathrm{cost}(P_t)$. Similarly we eliminate $\mathrm{cost}(P_t)$ using the second and the third bound. The approximation ratio we obtain is

$$\frac{3\sigma-2+(\sigma-1)\frac{2Y+1}{2^Z}}{2\sigma-1-(\sigma-1)\frac{2}{Y}}.$$

This expression converges to $\frac{3\sigma-2}{2\sigma-1}$ as Z and Y grow to infinity. It is clear that with the right choice of constant parameters Z and Y we will obtain a value within $\frac{3\sigma-2}{2\sigma-1}+\epsilon$ for any $\epsilon>0$. Polynomial running time follows from the fact, that Connect runs in a polynomial time. □

4 Edge Cost Increase

In this section we consider the reoptimization scenario of increasing the cost of one edge. We provide a polynomial-time 1.281-approximation algorithm for this reoptimization variant.

Recall that the *SMT* Edge Cost Increase problem takes as input two *SMT* instances (G,S) and (G',S) together with an optimal solution Opt for (G,S), where $G=(V,E,\mathrm{cost})$, $G'=(V,E,\mathrm{cost}')$ and cost coincides with cost′ on all the edges but one edge e^*, where $\mathrm{cost}(e^*)<\mathrm{cost}'(e^*)$. We will show that without loss of generality G and G' may be assumed to be metric. For a graph G, we call an edge $f\in E(G)$ *implied*, if its cost in the metric closure $\overline{G}$ is the cost of some path between its endpoints, and this path is not just a single edge f.

Remark 2. Without loss of generality we may assume that the input instance to *SMT* under edge cost increase consists of two *SMT* problem instances (G,S) and (G',S) and a solution Opt for (G,S), where $G=(V,E,\mathrm{cost})$ and $G'=(V,E,\mathrm{cost}')$ are metric and complete, Opt satisfies Observation 1 and for every $f\in \textsc{Opt}-e^*$ it holds that $\mathrm{cost}(f)=\mathrm{cost}'(f)$ whereas $\mathrm{cost}(e^*)<\mathrm{cost}'(e^*)$.

Proof. Assume G and G' are not metric and complete. Observe that Opt does not contain implied edges. Since cost and cost′ coincide on all the edges but e^*, for every $f\subseteq \textsc{Opt}-e^*$ it holds that $\overline{\mathrm{cost}}(f)=\overline{\mathrm{cost}'}(f)$ whereas $\overline{\mathrm{cost}}(e^*)<$

$\overline{\mathrm{cost}'}(e^*)$. We solve the reoptimization problem for the instance $(\overline{G}, S)$, $(\overline{G'}, S)$ and an optimal solution $\textsc{Opt}$ for $(\overline{G}, S)$. This instance satisfies Remark 2. Note that when transforming $\textsc{Opt}$ to its counterpart satisfying Observation 1 we preserve the invariant that at most one edge in $\textsc{Opt}$ increases its cost in $\overline{G'}$. The approximate solution for $(\overline{G'}, S)$ can easily be transformed to the solution for (G', S) with the same cost. □

Lemma 6. *For any constant $\epsilon > 0$, there is a polynomial-time $\left(\frac{2\sigma-1}{\sigma} + \epsilon\right)$-approximation algorithm for the SMT Edge Cost Increase.*

Proof. Let Z be a parameter. Let $S = S_1 \cup S_2$ be a partition of the terminals depending on to which tree in $\textsc{Opt} - e^*$ they belong. Let $\textsc{Opt}'$ be an optimal solution to (G', S). Consider $P^{(3)} \cup B_1 \cup B_2 \subseteq \textsc{Opt}'$ as in Lemma 3 for S_1 and S_2. Let $B_i^{(Z)}$, $i \in \{1, 2\}$, be the first Z levels of B_i starting from the root and let $T^{(Z)} = P^{(3)} \cup B_1^{(Z)} \cup B_2^{(Z)}$. For a tree T, let $f_1(T)$ be the cheapest edge connecting it with S_1 and $f_2(T)$ be the cheapest edge connecting it with S_2. The reoptimization algorithm guesses $T^{(Z)}$, and returns a better solution among $\textsc{Opt} - e^* \cup T^{(Z)} \cup f_1(T^{(Z)}) \cup f_2(T^{(Z)})$ and $\mathcal{S}_{\textsc{ApprSMT}}((G', S), T^{(Z)})$. By "guesses" we mean that it iterates over all trees T of size bounded by $2^{Z+1} + 3$. For each T, it computes $\textsc{Opt} - e^* \cup T \cup f_1(T) \cup f_2(T)$ and $\mathcal{S}_{\textsc{ApprSMT}}((G', S), T)$. It returns the best of the computed solutions.

Note that $\mathrm{cost}'(\textsc{Opt} - e^*) \le \mathrm{cost}'(\textsc{Opt}')$ because

$$\mathrm{cost}'(\textsc{Opt} - e^*) = \mathrm{cost}(\textsc{Opt} - e^*) \le \mathrm{cost}(\textsc{Opt}) \le \mathrm{cost}(\textsc{Opt}') \le \mathrm{cost}'(\textsc{Opt}').$$

Due to Lemma 3,

$$\mathrm{cost}'(f_1(T^{(Z)})), \mathrm{cost}'(f_2(T^{(Z)})) \le \frac{\mathrm{cost}'(\textsc{Opt}')}{2^{Z+1}}.$$

Therefore,

$$\begin{aligned}\mathrm{cost}'(\textsc{Sol}) &\le \mathrm{cost}'(\textsc{Opt} - e^* \cup T^{(Z)} \cup f_1(T^{(Z)}) \cup f_2(T^{(Z)})) \\ &\le (1 + \frac{1}{2^Z})\mathrm{cost}'(\textsc{Opt}') + \mathrm{cost}'(T^{(Z)}).\end{aligned}$$

On the other hand, by the power of Lemma 3,

$$\mathrm{cost}'(\mathcal{S}_{\textsc{ApprSMT}}((G', S), T^{(Z)})) \le \sigma \mathrm{cost}'(\textsc{Opt}') - (\sigma - 1)\mathrm{cost}'(T^{(Z)}),$$

where σ is the approximation ratio of an algorithm $\textsc{ApprSMT}$ for *SMT*. Again, we view these two bounds on the overall solution as functions of $\mathrm{cost}'(T^{(Z)})$, increasing and decreasing respectively. We compute the maximum value of the minimum of these two bounds by setting them equal. The resulting approximation ratio is

$$\frac{2\sigma - 1 + \frac{\sigma-1}{2^Z}}{\sigma}.$$

This ratio can be arbitrarily close to $\frac{2\sigma-1}{\sigma}$ with the right choice of constant parameter Z. □

5 Concluding Remarks

To conclude the paper, we present the state of the art on the *SMT* reoptimization in Table 1. An open question concerning the reoptimization variants of *SMT* studied here is the existence of a PTAS for them. In [16], the authors proved that there cannot be a PTAS for the modification of adding a vertex to the graph. They also provided a PTAS for all the modifications studied here if the input graph is β-metric for some constant $\beta < 1$. A PTAS for the metric case remains an interesting open problem.

Table 1. Different types of *SMT* modifications with the corresponding best up to date approximation ratios

The *SMT* problem under:			
Terminal		Edge cost	
addition	removal	decrease	increase
1.218 [22]	1.218 here	1.246 [5]	1.279 here

References

1. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the traveling salesman problem. Networks 42(3), 154–159 (2003)
2. Archetti, C., Luca, B., Speranza, M.G.: Reoptimizing the 0-1 knapsack problem. Technical Report 267, University of Brescia (2006)
3. Ausiello, G., Escoffier, B., Monnot, J., Paschos, V.T.: Reoptimization of Minimum and Maximum Traveling Salesman's Tours. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 196–207. Springer, Heidelberg (2006)
4. Bern, M.W., Plassmann, P.E.: The Steiner problem with edge lengths 1 and 2. Inf. Process. Lett. 32(4), 171–176 (1989)
5. Bilò, D., Böckenhauer, H.-J., Hromkovič, J., Královič, R., Mömke, T., Widmayer, P., Zych, A.: Reoptimization of Steiner Trees. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 258–269. Springer, Heidelberg (2008)
6. Bilò, D., Böckenhauer, H.-J., Komm, D., Králović, R., Mömke, T., Seibert, S., Zych, A.: Reoptimization of the shortest common superstring problem. Algorithmica 61(2), 227–251 (2011)
7. Bilò, D., Widmayer, P., Zych, A.: Reoptimization of Weighted Graph and Covering Problems. In: Bampis, E., Skutella, M. (eds.) WAOA 2008. LNCS, vol. 5426, pp. 201–213. Springer, Heidelberg (2009)
8. Böckenhauer, H.-J., Forlizzi, L., Hromkovič, J., Kneis, J., Kupke, J., Proietti, G., Widmayer, P.: Reusing Optimal TSP Solutions for Locally Modified Input Instances (Extended Abstract). In: Navarro, G., Bertossi, L., Kohayakawa, Y. (eds.) TCS 2006. IFIP, vol. 209, pp. 251–270. Springer, Boston (2006)
9. Böckenhauer, H.-J., Hromković, J., Králović, R., Mömke, T., Rossmanith, P.: Reoptimization of Steiner trees: Changing the terminal set, vol. 410, pp. 3428–3435. Elsevier Science Publishers Ltd. (August 2009)

10. Böckenhauer, H.-J., Hromkovič, J., Mömke, T., Widmayer, P.: On the Hardness of Reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 50–65. Springer, Heidelberg (2008)
11. Böckenhauer, H.-J., Komm, D.: Reoptimization of the Metric Deadline TSP. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 156–167. Springer, Heidelberg (2008)
12. Byrka, J., Grandoni, F., Rothvoß, T., Sanità, L.: An Improved LP-based Approximation for Steiner Tree. In: STOC 2010, Best Paper Award (2010)
13. Escoffier, B., Milanic, M., Paschos, V.T.: Simple and fast reoptimizations for the Steiner tree problem. Technical Report 2007-01, DIMACS (2007)
14. Escoffier, B., Milanič, M., Paschos, V.T.: Simple and fast reoptimizations for the Steiner tree problem 4(2), 86–94 (2009)
15. Escoffier, B., Ausiello, G., Bonifaci, V.: Complexity and Approximation in Reoptimization. In: Computability in Context: Computation and Logic in the Real World. Imperial College Press (2011)
16. Böckenhauer, H.-J., Freiermuth, K., Hromkovič, J., Mömke, T., Sprock, A., Steffen, B.: The Steiner Tree Reoptimization Problem with Sharpened Triangle Inequality. In: Calamoneri, T., Diaz, J. (eds.) CIAC 2010. LNCS, vol. 6078, pp. 180–191. Springer, Heidelberg (2010)
17. Mikhailyuk, V.A.: Reoptimization of set covering problems. Cybernetics and Sys. Anal. 46, 879–883 (2010)
18. Prömel, H.J., Steger, A.: The Steiner Tree Problem. Advanced Lectures in Mathematics. Friedr. Vieweg & Sohn, Braunschweig (2002)
19. Robins, G., Zelikovsky, A.: Improved Steiner tree approximation in graphs. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 770–779. ACM (2000)
20. Schäffter, M.W.: Scheduling with forbidden sets. Discrete Applied Mathematics 72(1-2), 155–166 (1997)
21. Zych, A.: Reoptimization of NP-hard problems. Ph.D. thesis, ETH Zürich
22. Zych, A., Bilò, D.: New reoptimization techniques applied to Steiner tree problem. Electronic Notes in Discrete Mathematics 37, 387–392 (2011); LAGOS 2011 - VI Latin-American Algorithms, Graphs and Optimization Symposium

Smoothed Complexity Theory*

Markus Bläser[1] and Bodo Manthey[2]

[1] Saarland University
mblaeser@cs.uni-saarland.de
[2] University of Twente
b.manthey@utwente.nl

Abstract. Smoothed analysis is a new way of analyzing algorithms introduced by Spielman and Teng (*J. ACM*, 2004). Classical methods like worst-case or average-case analysis have accompanying complexity classes, like P and Avg-P, respectively. While worst-case or average-case analysis give us a means to talk about the running time of a particular algorithm, complexity classes allows us to talk about the inherent difficulty of problems.

Smoothed analysis is a hybrid of worst-case and average-case analysis and compensates some of their drawbacks. Despite its success for the analysis of single algorithms and problems, there is no embedding of smoothed analysis into computational complexity theory, which is necessary to classify problems according to their intrinsic difficulty.

We propose a framework for smoothed complexity theory, define the relevant classes, and prove some first results.

1 Introduction

The goal of computational complexity theory is to classify computational problems according to their intrinsic difficulty. While the analysis of algorithms is concerned with analyzing, say, the running time of a particular algorithm, complexity theory rather analyses the amount of resources that all algorithms need at least to solve a given problem.

Classical complexity classes, like P, reflect worst-case analysis of algorithms. Worst-case analysis has been a success story: The bounds obtained are valid for every input of a given size, and, thus, we do not have to think about typical instances of our problem. If an algorithm has a good worst-case upper bound, then this is a very strong statement: The algorithm will perform well in practice.

However, some algorithms work well in practice despite having a provably high worst-case running time. The reason for this is that the worst-case running time can be dominated by a few pathological instances that rarely or never occur in practice. An alternative to worst-case analysis is average-case analysis. Many of the algorithms with poor worst-case but good practical performance have a good average running time. This means that the expected running time with instances drawn according to some fixed probability distribution is low.

* Supported by DFG research grant BL 511/7-1. A full version of this paper is available at http://arxiv.org/abs/1202.1936

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 198–209, 2012.

In complexity-theoretic terms, P is the class of all problems that can be solved with polynomial worst-case running time. In the same way, the class Avg-P is the class of all problems that have polynomial average-case running time. Average-case complexity theory studies the structural properties of average-case running time. Bogdanov and Trevisan give a comprehensive survey of average-case complexity [7].

While worst-case complexity has the drawback of being often pessimistic, the drawback of average-case analysis is that random instances have often very special properties with high probability. These properties of random instances distinguish them from typical instances. Since a random and a typical instance is not the same, a good average-case running time does not necessarily explain a good performance in practice. In order to get a more realistic performance measure, (and, in particular, to explain the speed of the simplex method), Spielman and Teng have proposed a new way to analyze algorithms called *smoothed analysis* [27]. In smoothed analysis, an adversary chooses an instance, and then this instance is subjected to a slight random perturbation. We can think of this perturbation as modeling measurement errors or random noise or the randomness introduced by taking, say, a random poll. The perturbation is controlled by some parameter ϕ, called the *perturbation parameter*. Spielman and Teng have proved that the simplex method has a running time that is polynomial in the size of the instance and the perturbation parameter [27]. Since then, the framework of smoothed analysis has been applied successfully to a variety of algorithms that have a good behavior in practice (and are therefore widely used) but whose worst-case running time indicates poor performance [1,2,5,12,13,16,23,26,29]. We refer to two recent surveys for a broader picture of smoothed analysis [22,28]. However, with only few exceptions [3,25], smoothed analysis has only been applied yet to single algorithms or single problems. Up to our knowledge, there is currently no attempt to formulate a smoothed complexity theory and, thus, to embed smoothed analysis into computational complexity.

This paper is an attempt to define a smoothed complexity theory, including notions of intractability, reducibility, and completeness. We define the class Smoothed-P (Section 2), which corresponds to problems that can be solved smoothed efficiently, we provide a notion of reducibility (Section 3), and define the class $\mathsf{Dist\text{-}NP}_{\mathrm{para}}$, which is a smoothed analogue of NP, and prove that it contains complete problems (Section 4). We continue with some basic observations (Section 5). We also add examples of problems in Smoothed-P (Sections 6 and 7) and discuss the relationship of smoothed complexity to semi-random models (Section 8). Finally, we conclude with a discussion of extension, shortcomings, and difficulties of our definitions (Section 9).

2 Smoothed Polynomial Time and Smoothed-P

2.1 Basic Definitions

In the first application of smoothed analysis to the simplex method [27], the strength of the perturbation has been controlled in terms of the standard deviation σ of the Gaussian perturbation. While this makes sense for numerical problems, this model cannot be used for general (discrete problems). A more general form of perturbation models has been introduced by Beier and Vöcking [2]: Instead of specifying an instance that is afterwards perturbed (which can also be viewed as the adversary specifying the mean of the probability distribution according to which the instances are drawn), the adversary specifies the whole probability distribution. Now the role of the standard deviation σ is taken over by the parameter ϕ, which is an upper bound for the maximum density of the probability distributions. For Gaussian perturbation, we have $\sigma = \Theta(1/\phi)$. Because we do not want to restrict our theory to numerical problems, we have decided to use the latter model.

Let us now define our model formally. Our perturbation models are families of distributions $\mathcal{D} = (D_{n,x,\phi})$. The length of x is n (so we could omit the index n but we keep it for clarity). Note that length does not necessarily mean bit length, but depends on the problem. For instance, it can be the number of vertices of the graph encoded by x. For every n, x, and ϕ, the support of the distribution $D_{n,x,\phi}$ should be contained in the set $\{0,1\}^{\leq \mathrm{poly}(n)}$. Let $S_{n,x} = \bigl\{y \mid D_{n,x,\phi}(y) > 0 \text{ for some } \phi\bigr\}$, and let $N_{n,x} = |S_{n,x}|$.

For all n, x, ϕ, and y, we demand $D_{n,x,\phi}(y) \leq \phi$. This controls the strength of the perturbation and restricts the adversary. We allow $\phi \in [1/N_{n,x}, 1]$. Furthermore, the values of ϕ are discretized, so that they can be described by at most $\mathrm{poly}(n)$ bits. The case $\phi = 1$ corresponds to the worst-case complexity; we can put all the mass on one string. The case $\phi = 1/N_{n,x}$ models the average case; here we usually have to put probability on an exponentially large set of strings. In general, the larger ϕ, the more powerful the adversary. We call such families $(D_{n,x,\phi})_{n,x,\phi}$ of probability distributions *parameterized families of distributions.*

Now we can specify what it means that an algorithm has smoothed polynomial running-time. The following definition can also be viewed as a discretized version of Beier and Vöcking's definition [3]. Note that we do not speak about expected running-time, but about expected running-time to some power ε. This is because the notion of expected running-time is not robust with respect to, e.g., quadratic slowdown. The corresponding definition for average-case complexity is due to Levin [20]. We refer to Bogdanov and Trevisan [7] for a thorough discussion of this issue.

Definition 2.1. *An algorithm A has* smoothed polynomial running time *with respect to the family $\mathcal{D}$ if there exists an $\varepsilon > 0$ such that, for all n, x, and ϕ, we have $\mathbb{E}_{y \sim D_{n,x,\phi}}\bigl(t_A(y; n, \phi)^{\varepsilon}\bigr) = O\bigl(n \cdot N_{n,x} \cdot \phi\bigr)$.*

This definition implies that (average-)polynomial time is only required if we have $\phi = O(\mathrm{poly}(n)/N_{n,x})$. This seems to be quite generous at first glance, but it is in accordance with, e.g., Spielman and Teng's analysis of the simplex method [27] or Beier and Vöcking's analysis of integer programs [3]; they achieve polynomial time running time only if they perturb all but at most $O(\log n)$ digits: If we perturb a number with, say, a Gaussian of standard deviation $\sigma = 1/\mathrm{poly}(n)$, then we expect that the $O(\log n)$ most significant bits remain untouched, but the less significant bits are random.

In average-case complexity, one considers not decision problems alone, but decision problems together with a probability distribution. The smoothed analogue of this is that we consider tuples $(L, \mathcal{D})$, where $L \subseteq \{0,1\}^*$ is a decision problem and $\mathcal{D}$ is a parameterized family of distributions. We call such problems *parameterized distributional problems.* The notion of smoothed polynomial running-time (Definition 2.1) allows us to define what it means for a parameterized distributional problem to have polynomial smoothed complexity.

Definition 2.2. Smoothed-P *is the class of all* $(L, \mathcal{D})$ *such that there is a deterministic algorithm* A *with smoothed polynomial running time that decides* L*.*

We start with an alternative characterization of smoothed polynomial time as it is known for the average case: an algorithm has smoothed polynomial running-time if and only if its running-time has polynomially decreasing tail bounds.

Theorem 2.3. *An algorithm* A *has smoothed polynomial running time if and only if there is an* $\varepsilon > 0$ *and a polynomial* p *such that for all* n*,* x*,* ϕ*, and* t*,* $\Pr_{y \sim D_{n,x,\phi}}[t_A(y; n, \phi) \geq t] \leq \frac{p(n)}{t^\varepsilon} \cdot N_{n,x} \cdot \phi$*.*

2.2 Heuristic Schemes

A different way to think about efficiency in the smoothed setting is via so-called heuristic schemes. This notion comes from average-case complexity [7], but can be adapted to our smoothed setting. The notion of a heuristic scheme comes from the observation that, in practice, we might only be able to run our algorithm for a polynomial number of steps. If the algorithms does not succeed within this time bound, then it "fails", i.e., it does not solve the given instance. The failure probability decreases polynomially with the running time that we allow. The following definition captures this.

Definition 2.4. *Let* $(L, \mathcal{D})$ *be a smoothed distributional problem. An algorithm* A *is an errorless heuristic scheme for* $(L, \mathcal{D})$ *if there is a polynomial* q *such that*

1. *For every* n*, every* x*, every* ϕ*, every* $\delta > 0$*, and every* $y \in \operatorname{supp} D_{n,x,\phi}$*, we have* $A(y; n, \phi, \delta)$ *outputs either* $L(y)$ *or* $\perp$*.*
2. *For every* n*, every* x*, every* ϕ*, every* $\delta > 0$*, and every* $y \in \operatorname{supp} D_{n,x,\phi}$*, we have* $t_A(y; n, \delta) \leq q(n, N_{n,x}\phi, 1/\delta)$*.*
3. *For every* n*,* x*,* ϕ*,* $\delta > 0$*, and* $y \in \operatorname{supp} D_{n,x,\phi}$*,* $\Pr_{y \sim D_{n,x,\phi}}[A(y; n, \phi, \delta) = \perp] \leq \delta$*.*

Theorem 2.5. $(L, \mathcal{D}) \in$ Smoothed-P *if and only if* $(L, \mathcal{D})$ *has an errorless heuristic scheme.*

2.3 Alternative Definition: Bounded Moments

At first glance, one might be tempted to use "expected running time" for the definition of Avg-P and Smoothed-P. However, as mentioned above, simply using the expected running time does not yield a robust measure. This is the reason why the expected value of the running time raised to some (small) constant power is used. Röglin and Teng [24, Theorem 6.2] have shown that for integer programming (more precisely, for binary integer programs with a linear objective function), the expected value indeed provides a robust measure. They have proved that a binary optimization problem can be solved in expected polynomial time if and only if it can be solved in worst-case pseudo-polynomial time. The reason for this is that all finite moments of the Pareto curve are polynomially bounded. Thus, a polynomial slowdown does not cause the expected running time to jump from polynomial to exponential.

As far as we are aware, this phenomenon, i.e., the case that all finite moments have to be bounded by a polynomial, has not been studied yet in average-case complexity. Thus, for completeness, we define the corresponding average-case and smoothed complexity classes as an alternative to $\mathsf{Avg\text{-}P}$ and $\mathsf{Smoothed\text{-}P}$.

Definition 2.6. *1. An algorithm has* robust smoothed polynomial running time *with respect to $\mathcal{D}$ if, for all fixed $\varepsilon > 0$ and for every n, x, and ϕ, we have $\mathbb{E}_{y \sim D_{n,x,\phi}}\big(t_A(y; n, \phi)^\varepsilon\big) = O\big(n \cdot N_{n,x} \cdot \phi\big)$. $\mathsf{Smoothed\text{-}PBM}$ is the class of all $(L, \mathcal{D})$ for which there exists a deterministic algorithm with robust smoothed polynomial running time. (The "PBM" stands for "polynomially bounded moments".)*

2. An algorithm A has robust average polynomial running time *with respect to $\mathcal{D}$ if, for all fixed $\varepsilon > 0$ and for all n, we have $\mathbb{E}_{y \sim D_n}\big(t_A(y)^\varepsilon\big) = O(n)$. $\mathsf{Avg\text{-}PBM}$ contains all $(L, \mathcal{D})$ for which there exists a deterministic algorithm with robust smoothed polynomial running time.*

From the definition, we immediately get $\mathsf{Smoothed\text{-}PBM} \subseteq \mathsf{Smoothed\text{-}P}$ and $\mathsf{Avg\text{-}PBM} \subseteq \mathsf{Avg\text{-}P}$. Moreover, if $L \in \mathsf{P}$, then L together with any family of distributions is also in $\mathsf{Smoothed\text{-}P}$ and $\mathsf{Avg\text{-}P}$ and also in $\mathsf{Smoothed\text{-}PBM}$ and $\mathsf{Avg\text{-}PBM}$. From Röglin and Teng's result [24], one might suspect $\mathsf{Avg\text{-}P} = \mathsf{Avg\text{-}PBM}$ and $\mathsf{Smoothed\text{-}P} = \mathsf{Smoothed\text{-}PBM}$, but this does not hold.

Theorem 2.7. $\mathsf{Avg\text{-}PBM} \subsetneq \mathsf{Avg\text{-}P}$ *and* $\mathsf{Smoothed\text{-}PBM} \subsetneq \mathsf{Smoothed\text{-}P}$.

3 Disjoint Supports and Reducibility

The same given input y can appear with very high and with very low probability at the same time. What sounds like a contradiction has an easy explanation: $D_{n,x,\phi}(y)$ can be large whereas $D_{n,x',\phi}(y)$ for some $x' \neq x$ is small. But if we only see y, we do not know whether x or x' was perturbed. This causes some problems when one wants to develop a notion of reduction and completeness.

For a parameterized distributional problem $(L, \mathcal{D})$, let

$$L_{\mathrm{ds}} = \{\langle x, y\rangle \mid y \in L \text{ and } |y| \leq \mathrm{poly}(|x|)\}.$$

The length of $|y|$ is bounded by the same polynomial that bounds the length of the strings in any $\operatorname{supp} D_{n,x,\phi}$. We will interpret a pair $\langle x, y\rangle$ as "y was drawn according to $D_{n,x,\phi}$". With the notion of L_{ds}, we can now define a reducibility between parameterized distributional problems. We stress that, although the definition below involves L_{ds} and L'_{ds}, the reduction is defined for pairs L and L' and neither of the two is required to be a disjoint-support language. This means that, for $(L, \mathcal{D})$, the supports of $D_{n,x,\phi}$ for different x may intersect. And the same is allowed for $(L', \mathcal{D}')$.

Definition 3.1. *Let $(L, \mathcal{D})$ and $(L', \mathcal{D})$ be two parameterized distributional problems. $(L, \mathcal{D})$ reduces to $(L', \mathcal{D}')$ (denoted by "$(L, \mathcal{D}) \leq_{\text{smoothed}} (L', \mathcal{D}')$") if there is a polynomial time computable function f such that for every n, every x, every ϕ and every $y \in \operatorname{supp} D_{n,x,\phi}$ the following holds:*

1. *$\langle x, y\rangle \in L_{\mathrm{ds}}$ if and only if $f(\langle x, y\rangle; n, \phi) \in L'_{\mathrm{ds}}$.*
2. *There exist polynomials p and m such that, for every n, x, and ϕ and every $y' \in \operatorname{supp} D'_{m(n), f_1(\langle x,y\rangle; n, \phi), \phi}$, we have*

$$\textstyle\sum_{y: f_2(\langle x,y\rangle;n,\phi)=y'} D_{n,x,\phi}(y) \leq p(n) D_{m(n), f_1(\langle x,y\rangle;n,\phi),\phi}(y'),$$

where $f(\langle x, y\rangle; n, \phi) = \langle f_1(\langle x, y\rangle; n, \phi), f_2(\langle x, y\rangle; n, \phi)\rangle$.

Remark 3.2. We could also allow that ϕ on the right-hand side is polynomially transformed. However, we currently do not see how to benefit from this.

It is easy to see that $\leq_{\text{smoothed}}$ is transitive. Ideally, Smoothed-P should be closed under this type of reductions. However, we can only show this for the related class of problems with disjoint support.

Definition 3.3. Smoothed-P$_{\mathrm{ds}}$ *is the set of all distributional problems with disjoint supports such that there is an algorithm A for L_{ds} with smoothed polynomial running time. (Here, the running time on $\langle x, y\rangle$ is defined in the same way as in Definition 2.1. Since $|y| \leq \operatorname{poly}(|x|)$ for a pair $\langle x, y\rangle \in L_{\mathrm{ds}}$, we can as well measure the running time in $|x|$.)*

Theorem 3.4. *If $(L, \mathcal{D}) \leq_{\text{smoothed}} (L', \mathcal{D}')$ and $(L'_{\mathrm{ds}}, \mathcal{D}') \in$ Smoothed-P$_{\mathrm{ds}}$, then $(L_{\mathrm{ds}}, \mathcal{D}) \in$ Smoothed-P$_{\mathrm{ds}}$.*

With the definition of disjoint support problems, a begging question is how the complexity of L and L_{ds} are related. It is obvious that $(L, \mathcal{D}) \in$ Smoothed-P implies $(L_{\mathrm{ds}}, \mathcal{D}) \in$ Smoothed-P$_{\mathrm{ds}}$. However, the converse is not so obvious. The difference between L and L_{ds} is that for L_{ds}, we get the x from which the input y was drawn. While this extra information does not seem to be helpful at a first glance, we can potentially use it to extract randomness from it. So this question is closely related to the problem of derandomization.

But there is an important subclass of problems in Smoothed-P$_{\mathrm{ds}}$ whose counterparts are in Smoothed-P, namely those which have an oblivious algorithm with smoothed polynomial running time. We call an algorithm (or heuristic scheme) for some problem with disjoint supports *oblivious* if the running time on $\langle x, y\rangle$ does not depend on x (up to constant factors). Let Smoothed-P$_{\mathrm{ds}}^{\mathrm{obl}}$ be the resulting subset of problems in Smoothed-P$_{\mathrm{ds}}$ that have such an oblivious algorithm with smoothed polynomial running time.

Theorem 3.5. *For any parameterized problem* $(L, \mathcal{D})$, $(L, \mathcal{D}) \in \mathsf{Smoothed\text{-}P}$ *if and only if* $(L_{\mathrm{ds}}, \mathcal{D}) \in \mathsf{Smoothed\text{-}P}^{\mathrm{obl}}_{\mathrm{ds}}$.

Note that almost all algorithms, for which a smoothed analysis has been carried out, do not know the x from which y was drawn; in particular, there is an oblivious algorithm for them. Thus, a begging questions is if there is a problem $(L, \mathcal{D}) \notin \mathsf{Smoothed\text{-}P}$ but $(L_{\mathrm{ds}}, \mathcal{D}) \in \mathsf{Smoothed\text{-}P}_{\mathrm{ds}}$.

Note that in L_{ds}, each y is paired with *every* x, so there is no possibility to encode information by omitting some pairs. This prohibits attempts for constructing such a problem like considering pairs $\langle x, f(x) \rangle$ where f is some one-way function. However, a pair $\langle x, y \rangle$ contains randomness that one could extract. On the other hand, for the classes Smoothed-BPP or Smoothed-P/poly, which can be defined in the obvious way, it seems plausible that knowing x does not seem to help.

4 Parameterized Distributional NP

In this section, we define the smoothed analogue of the worst-case class NP and the average-case class DistNP [18, 20]. First, we have to restrict ourself to "natural" distributions. This rules out, for instance, probability distributions based on Kolmogorov complexity that (the *universal distribution*), under which worst-case complexity equals average-case complexity for all problems [21]. We transfer the notion of computable ensembles to smoothed complexity, which allows us to define the smoothed analogue of NP and DistNP.

Definition 4.1. *A parameterized family of distributions is in* $\mathsf{PComp}_{\mathrm{para}}$ *if the cumulative probability* $F_{D_{n,x,\phi}} = \sum_{z \leq x} D_{n,x,\phi}$ *can be computed in polynomial time (given* n, x *and* ϕ *in binary).*

Definition 4.2. $\mathsf{Dist\text{-}NP}_{\mathrm{para}} = \{(L, \mathcal{D}) \mid L \in \mathsf{NP}$ *and* $\mathcal{D} \in \mathsf{PComp}_{\mathrm{para}}\}$.

Bounded halting – given a Turing machine, an input, and a running-time bound, does the Turing machine halt on this input within the given time bound – is complete for $\mathsf{Dist\text{-}NP}_{\mathrm{para}}$. Bounded halting is the canonical NP-complete language, and it has been the first problem that has been shown to be Avg-P-complete [20]. Formally, let

$$\mathtt{BH} = \{\langle g, x, 1^t \rangle \mid \text{NTM with Gödel number } g \text{ accepts } x \text{ within } t \text{ steps}\}.$$

For a specific parameterized family $U^{\mathtt{BH}}$ of distributions, we can prove the following theorem.

Theorem 4.3. $(\mathtt{BH}, U^{\mathtt{BH}})$ *is* $\mathsf{Dist\text{-}NP}_{\mathrm{para}}$*-complete for some* $U^{\mathtt{BH}} \in \mathsf{PComp}_{\mathrm{para}}$.

The original DistNP-complete problem by Levin [20] was `Tiling`: An instance of the problem consists of a finite set T of square tiles, a positive integer t, and a sequence $s = (s_1, \ldots, s_n)$ for some $n \leq t$ such that s_i matches s_{i+1} (the right side of s_i equals the left side of s_{i+1}). The question is whether S can be extended to tile an $n \times n$ square using tiles from T. Again, we need a special family $U^{\mathtt{Tiling}}$ of distributions.

Theorem 4.4. $(\texttt{Tiling}, U^{\texttt{Tiling}})$ *is* $\mathsf{Dist\text{-}NP}_{\text{para}}$*-complete for some* $U^{\texttt{Tiling}} \in \mathsf{PComp}_{\text{para}}$ *under polynomial-time smoothed reductions.*

5 Basic Relations to Worst-Case Complexity

In this section, we collect some simple facts about $\mathsf{Smoothed\text{-}P}$ and $\mathsf{Dist\text{-}NP}_{\text{para}}$ and their relationship to their worst-case and average-case counterparts.

Theorem 5.1. *If* $L \in \mathsf{P}$*, then* $(L, \mathcal{D}) \in \mathsf{Smoothed\text{-}P}$ *for any* $\mathcal{D}$*. If* $(L, \mathcal{D}) \in \mathsf{Smoothed\text{-}P}$ *with* $\mathcal{D} = (D_{n,x,\phi})_{n,x,\phi}$*, then* $(L, (D_{n,x_n,\phi})_n) \in \mathsf{Avg\text{-}P}$ *for* $\phi = O(\text{poly}(n)/N_{n,x})$ *and every sequence* $(x_n)_n$ *of strings with* $|x_n| \leq \text{poly}(n)$*.*

It is known that $\mathsf{DistNP} \subseteq \mathsf{Avg\text{-}P}$ implies $\mathsf{NE} = \mathsf{E}$ [4]. This can be transferred to smoothed complexity.

Theorem 5.2. *If* $\mathsf{Dist\text{-}NP}_{\text{para}} \subseteq \mathsf{Smoothed\text{-}P}$*, then* $\mathsf{NE} = \mathsf{E}$*.*

6 Tractability 1: Integer Programming

Now we deal with tractable – in the sense of smoothed complexity – optimization problems: We show that if a binary integer linear program can be solved in pseudo-polynomial time, then the corresponding decision problem belongs to $\mathsf{Smoothed\text{-}P}$. This result is similar to Beier and Vöckings characterization [3]: Binary optimization problems have smoothed polynomial complexity (with respect to continuous distributions) if and only if they can be solved in randomized pseudo-polynomial time.

A binary optimization problem is an optimization problem of the form "maximize $c^T x$ subject to $w_i^T x \leq t_i$ for $i \in [k]$ and $x \in S \subseteq \{0,1\}^n$. The set S should be viewed as containing the "structure" of the problem. The simplest case is $k = 1$ and $S = \{0,1\}^n$; then the binary program above represents the knapsack problem. We assume that S is adversarial (i.e., non-random). Since we deal with decision problems in this paper rather than with optimization problems, we use the standard approach and introduce a threshold for the objective function. This means that the optimization problem becomes the question whether there is an $x \in S$ that fulfills $c^T x \geq b$ as well as $w_i^T x \leq t_i$ for all $i \in \{1, \ldots, k\}$. In the following, we treat the budget constraint $c^T x \geq b$ as an additional linear constraint for simplicity. We call this type of problems *binary decision problems.*

Let us now describe the perturbation model. For ease of presentation, we assume that we have just one linear constraint (whose coefficients will be perturbed) and everything else is encoded in the set S. The coefficients of the left-hand sides of the constraints are n-bit binary numbers. We do not make any assumption about the probability distribution of any single coefficient. Instead, our result holds for any family of probability distribution that fulfills the following properties: $w_1, \ldots, w_n$ are drawn according to independent distributions. The set S and the threshold t are part of the input and not subject to randomness. Thus, $N_{n,(S,w,t)} = 2^{n^2}$ for any instance (S, w, t) of size n. We assume that S can be encoded by a polynomially long string. Since $N_{n,(S,w)} = 2^{n^2}$, the perturbation parameter ϕ can vary between 2^{-n^2} (for the average case) and 1 (for the worst case).

Theorem 6.1. *If a binary decision problem can be solved in pseudo-polynomial time, then it is in* Smoothed-P.

Beier and Vöcking [3] have proved that (randomized) pseudo-polynomiality and smoothed polynomiality are equivalent. The reason why we do not get a similar result is as follows: Our "joint density" for all coefficients is bounded by ϕ, and the density of a single coefficient is bounded by $\phi^{1/n}$. In contrast, in the continuous version, the joint density is bounded by ϕ^n while a single coefficient has a density bounded by ϕ. However, our goal is to devise a general theory for arbitrary decision problems. This theory should include integer optimization, but it should not be restricted to integer optimization. The problem is that generalizing the concept of one distribution bounded by ϕ for each coefficient to arbitrary problems involves knowledge about the instances and the structure of the specific problems. This knowledge, however, is not available if we want to speak about classes of decision problems as in classical complexity theory.

7 Tractability 2: Graphs and Formulas

7.1 Graph Coloring and Smoothed Extension of $G_{n,p}$

The perturbation model that we choose is the *smoothed extension of* $G_{n,p}$ [28]: Given an adversarial graph $G = (V, E)$ and an $\varepsilon \in (0, 1/2]$, we obtain a new graph $G' = (V, E')$ on the same set of vertices by "flipping" each (non-)edge of G independently with a probability of ε. This means the following: If $e = \{u, v\} \in E$, then e is contained in E' with a probability of $1 - \varepsilon$. If $e = \{u, v\} \notin E$, then $\Pr(e \in E') = \varepsilon$. Transferred to our framework, this means the following: We represent a graph G on n vertices as a binary string of length $\binom{n}{2}$, and we have $N_{n,G} = 2^{\binom{n}{2}}$. The flip probability ε depends on ϕ: We choose $\varepsilon \leq 1/2$ such that $(1-\varepsilon)^{\binom{n}{2}} = \phi$. (For $\phi = 2^{-\binom{n}{2}} = 1/N_{n,G}$, we have a fully random graph with edge probabilities of 1/2. For $\phi = 1$, we have $\varepsilon = 0$, thus the worst case.)

k-`Coloring` is the decision problem whether the vertices of a graph can be colored with k colors such that no pair of adjacent vertices get the same color. k-`Coloring` is NP-complete for any $k \geq 3$ [17, GT 4].

Theorem 7.1. *For any* $k \in \mathbb{N}$, k-`Coloring` $\in$ Smoothed-P.

Remark 7.2. Bohman et al. [8] and Krivelevich et al. [19] consider a slightly different model for perturbing graphs: Given an adversarial graph, we add random edges to the graph to obtain our actual instance. No edges are removed.

They analyze the probability that the random graph thus obtained is guaranteed to contain a given subgraph H. By choosing H to be a clique of size $k+1$ and using a proof similar to Theorem 7.1's, we obtain that k-`Coloring` $\in$ Smoothed-P also with respect to this perturbation model.

7.2 Unsatisfiability and Smoothed-RP

Feige [14] and Coja-Oghlan et al. [11] have considered the following model: We are given a (relatively dense) adversarial Boolean k-CNF formula. Then we obtain our instance by negating each literal with a small probability. It is proved that such smoothed formulas are likely to be unsatisfiable, and that their unsatisfiability can be proved efficiently. However, their algorithms are randomized, thus we do not get a result that $k\mathsf{UNSAT}$ (this means that unsatisfiability problem for k-CNF formulas) for dense instances belongs to $\mathsf{Smoothed\text{-}P}$. However, it shows that $k\mathsf{UNSAT}$ for dense instance belongs to $\mathsf{Smoothed\text{-}RP}$, where $\mathsf{Smoothed\text{-}RP}$ is the smoothed analogue of RP: A pair $(L, \mathcal{D})$ is in $\mathsf{Smoothed\text{-}RP}$ if there is a randomized polynomial algorithm A with the following properties:

1. For all $x \notin L$, A outputs “no”. (This property is independent of the perturbation.)
2. For all $x \in L$, A outputs “yes” with a probability of at least 1/2. (This property is also independent of the perturbation.)
3. A has smoothed polynomial running time with respect to $\mathcal{D}$. (This property is independent of the internal randomness of A.)

Now, let $k\mathsf{UNSAT}_\beta$ be $k\mathsf{UNSAT}$ restricted to instances with at least βn clauses, where n denotes the number of variables. Let ε be the probability that a particular literal is negated. Feige [14] has presented a polynomial-time algorithm with the following property: If $\beta = \Omega(\sqrt{n \log\log n}/\varepsilon^2)$ and the perturbed instance of $k\mathsf{UNSAT}_\beta$ is unsatisfiable, which it is with high probability, then his algorithm proves that the formula is unsatisfiable with a probability of at least $1 - 2^{\Omega(-n)}$. The following result is a straightforward consequence.

Theorem 7.3. *$k\mathsf{UNSAT}_\beta \in \mathsf{Smoothed\text{-}RP}$ for $\beta = \Omega(\sqrt{n \log\log n})$.*

8 Smoothed Analysis vs. Semi-random Models

Semi-random models for graphs and formulas exist even longer than smoothed analysis and can be considered as precursors to smoothed analysis. The basic concept is as follows: Some instance is created randomly that possesses a particular property. This property can, for instance, be that the graph is k-colorable. After that, the adversary is allowed to modify the instance without destroying the property. For instance, the adversary can be allowed to add arbitrary edges between the different color classes. Problems that have been considered in this model or variants thereof are independent set [15], graph coloring [6, 9, 15], or finding sparse induced subgraphs [10]. However, we remark that these results do not easily fit into a theory of smoothed analysis. The reason is that in these semi-random models, we first have the random instance, which is then altered by the adversary. This is in contrast to smoothed analysis in general and our smoothed complexity theory in particular, where we the adversarial decisions come before the randomness is applied.

9 Discussion

Our framework has many of the characteristics that one would expect. We have reductions and complete problems and they work in the way one expects them to work. To define reductions, we have to use the concept of disjoint supports. It seems to be essential that we know the original instance x that the actual instance y was drawn from to obtain proper domination. Although this is somewhat unconventional, we believe that this is the right way to define reductions in the smoothed setting. The reason is that otherwise, we do not know the probabilities of the instances, which we need in order to apply the compression function. The compression function, in turn, seems to be crucial to prove hardness results. Still, an open question is whether a notion of reducibility can be defined that circumvents these problems. Moreover, many of the positive results from smoothed analysis can be cast in our framework, like it is done in Sections 6 and 7.

Many positive results in the literature state their bounds in the number of “entities” (like number of nodes, number of coefficients) of the instance. However, in complexity theory, we measure bounds in the length (number of symbols) of the input in order to get a theory for arbitrary problems, not only for problems of a specific type. To state bounds in terms of bit length makes things less tight, for instance the reverse direction of integer programming does not work. But still, we think it is more important and useful to use the usual notion of input length such that smoothed complexity fits with average-case and worst-case complexity.

We hope that the present work will stimulate further research in smoothed complexity theory in order to get a deeper understanding of the theory behind smoothed analysis.

References

1. Arthur, D., Manthey, B., Röglin, H.: Smoothed analysis of the k-means method. J. ACM 58(5) (2011)
2. Beier, R., Vöcking, B.: Random knapsack in expected polynomial time. J. Comput. System Sci. 69(3), 306–329 (2004)
3. Beier, R., Vöcking, B.: Typical properties of winners and losers in discrete optimization. SIAM J. Comput. 35(4), 855–881 (2006)
4. Ben-David, S., Chor, B., Goldreich, O., Luby, M.: On the theory of average case complexity. J. Comput. System Sci. 44(2), 193–219 (1992)
5. Bläser, M., Manthey, B., Rao, B.V.R.: Smoothed analysis of partitioning algorithms for Euclidean functionals. Algorithmica (to appear)
6. Blum, A.L., Spencer, J.: Coloring random and semi-random k-colorable graphs. J. Algorithms 19(2), 204–234 (1995)
7. Bogdanov, A., Trevisan, L.: Average-case complexity. Foundations and Trends in Theoret. Comput. Sci. 2(1), 1–106 (2006)
8. Bohman, T., Frieze, A.M., Krivelevich, M., Martin, R.: Adding random edges to dense graphs. Random Struct. Algorithms 24(2), 105–117 (2004)
9. Coja-Oghlan, A.: Colouring semirandom graphs. Combin. Probab. Comput. 16(4), 515–552 (2007)

10. Coja-Oghlan, A.: Solving NP-hard semirandom graph problems in polynomial expected time. J. Algorithms 62(1), 19–46 (2007)
11. Coja-Oghlan, A., Feige, U., Frieze, A.M., Krivelevich, M., Vilenchik, D.: On smoothed k-CNF formulas and the Walksat algorithm. In: Proc. 20th Ann. Symp. on Discrete Algorithms (SODA), pp. 451–460. SIAM (2009)
12. Damerow, V., Manthey, B., Meyer auf der Heide, F., Räcke, H., Scheideler, C., Sohler, C., Tantau, T.: Smoothed analysis of left-to-right maxima with applications. ACM Trans. Algorithms 8(3), article 30
13. Englert, M., Röglin, H., Vöcking, B.: Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. In: Proc. 18th Ann. Symp. on Discrete Algorithms (SODA), pp. 1295–1304. SIAM (2007)
14. Feige, U.: Refuting smoothed 3CNF formulas. In: Proc. 48th Ann. Symp. on Foundations of Computer Science (FOCS), pp. 407–417. IEEE (2007)
15. Feige, U., Kilian, J.: Heuristics for semirandom graph problems. J. Comput. System Sci. 63(4), 639–671 (2001)
16. Fouz, M., Kufleitner, M., Manthey, B., Zeini Jahromi, N.: On smoothed analysis of quicksort and Hoare's find. Algorithmica 62(3-4), 879–905 (2012)
17. Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Company (1979)
18. Gurevich, Y.: Average case completeness. J. Comput. System Sci. 42(3), 346–398 (1991)
19. Krivelevich, M., Sudakov, B., Tetali, P.: On smoothed analysis in dense graphs and formulas. Random Struct. Algorithms 29(2), 180–193 (2006)
20. Levin, L.A.: Average case complete problems. SIAM J. Comput. 15(1), 285–286 (1986)
21. Li, M., Vitányi, P.M.B.: Average case complexity under the universal distribution equals worst-case complexity. Inform. Process. Lett. 42(3), 145–149 (1992)
22. Manthey, B., Röglin, H.: Smoothed analysis: Analysis of algorithms beyond worst case. it – Information Technology 53(6) (2011)
23. Moitra, A., O'Donnell, R.: Pareto optimal solutions for smoothed analysts. In: Proc. 43rd Ann. Symp. on Theory of Computing (STOC), pp. 225–234. ACM (2011)
24. Röglin, H., Teng, S.-H.: Smoothed analysis of multiobjective optimization. In: Proc. 50th Ann. Symp. on Foundations of Computer Science (FOCS), pp. 681–690. IEEE (2009)
25. Röglin, H., Vöcking, B.: Smoothed analysis of integer programming. Math. Prog. 110(1), 21–56 (2007)
26. Spielman, D.A., Teng, S.-H.: Smoothed analysis of termination of linear programming algorithms. Math. Prog. 97(1-2), 375–404 (2003)
27. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. J. ACM 51(3), 385–463 (2004)
28. Spielman, D.A., Teng, S.-H.: Smoothed analysis: An attempt to explain the behavior of algorithms in practice. Commun. ACM 52(10), 76–84 (2009)
29. Vershynin, R.: Beyond Hirsch conjecture: Walks on random polytopes and smoothed complexity of the simplex method. SIAM J. Comput. 39(2), 646–678 (2009)

Abelian Pattern Avoidance in Partial Words*

Francine Blanchet-Sadri[1] and Sean Simmons[2]

[1] Department of Computer Science, University of North Carolina,
P.O. Box 26170, Greensboro, NC 27402–6170, USA
[2] Department of Mathematics, Massachusetts Institute of Technology,
77 Massachusetts Avenue, Cambridge, MA 02139–4307, USA

Abstract. Pattern avoidance is an important topic in combinatorics on words which dates back to the beginning of the twentieth century when Thue constructed an infinite word over a ternary alphabet that avoids squares, i.e., a word with no two adjacent identical factors. This result finds applications in various algebraic contexts where more general patterns than squares are considered. On the other hand, Erdős raised the question as to whether there exists an infinite word that avoids abelian squares, i.e., a word with no two adjacent factors being permutations of one another. Although this question was answered affirmately years later, knowledge of abelian pattern avoidance is rather limited. Recently, (abelian) pattern avoidance was initiated in the more general framework of partial words, which allow for undefined positions called holes. Here, we investigate conditions for a pattern to be abelian avoidable by a partial word with finitely or infinitely many holes.

1 Introduction

Combinatorics on *(full) words*, or sequences of letters over a finite alphabet, goes back to the work of the mathematician, Thue, at the beginning of the twentieth century [14]. The interest in this topic has been increasing since it finds applications in various research areas of mathematics, computer science, biology, and physics where the data can be easily represented as words over some alphabet. Motivated by molecular biology of nucleic acids, *partial words*, or sequences that may have some "do not know symbols", also called "holes", were introduced by Berstel and Boasson in [1], and have been extensively studied since (see [2] for instance). For example, $cca\diamond ctcg\diamond ccctc$ is a partial word with two holes, represented by the two $\diamond$'s, over the DNA alphabet $\{a, c, g, t\}$ (the $\diamond$'s are *compatible* with, or match, every letter of the alphabet). Partial words are interesting from a

* This material is based upon work supported by the National Science Foundation under Grant Nos. DMS–0754154 and DMS–1060775. The Department of Defense is gratefully acknowledged. The authors thank Dimin Xu for valuable comments and suggestions. A research assignment from the University of North Carolina at Greensboro for the first author is gratefully acknowledged. Some of this assignment was spent at the LIAFA: Laboratoire d'Informatique Algorithmique: Fondements et Applications of Université Paris 7-Denis Diderot, Paris, France.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 210–221, 2012.

theoretical point of view as they approximate full words, but also from a practical point of view. Such pratical uses occur for instance in bio-inspired computing where they have been considered for identifying good encodings for DNA computations [12]. More specifically, a basic issue in DNA computing is to find strands, which are used as codewords, that should not form so-called secondary structures (a sequence has such a structure if there is some kind of repetition in it). The combinatorial concept of repetition-freeness, or repetition avoidance, has been proposed to exclude such formation. Requiring a big Hamming distance between the strands has also been proposed. So partial words can be used as codewords, the latter requirement being provided through the compatibility relation. They offer a more powerful and realistic model than full words due to the errors caused by the evolutionary processes of deletion, insertion, and mutation in biological sequences. A deeper understanding of avoidance of patterns such as repetitions in the framework of partial words can thus be exploited.

In the context of full words, Thue showed that arbitrarily long non-repetitive words, or those that avoid repetitions, can be constructed with only three letters. The importance of this result has been seen through several rediscoveries that come from applications in ergodic theory, formal language theory, group theory, universal algebra, etc. In a recent paper, Currie [8] reviews results on pattern avoidance, and discusses a number of open problems on words avoiding squares and more general patterns which are words over an alphabet of *variables*, denoted by $\alpha, \beta, \gamma, \ldots$. Squares, for instance, are represented by the unary pattern $\alpha^2 = \alpha\alpha$, which is a power of a single variable α.

Erdős in 1961 initiated abelian pattern avoidance by raising the question whether abelian squares can be avoided [10]. A word contains an abelian square if it has a factor uv, where u is a permutation of v or u is a rearrangement of the letters of v. For example, $w = babcacb$ contains the abelian square $abcacb$ even though w is square-free. More generally, if $p = \alpha_0 \cdots \alpha_{n-1}$, where the α_i's are variables, a word *w meets p in the abelian sense* if w contains a factor $u_0 \cdots u_{n-1}$ where u_i is a permutation of u_j, whenever $\alpha_i = \alpha_j$; otherwise, *w avoids p in the abelian sense.* In 1979, Dekking showed that abelian cubes can be avoided over three letters, and two letters are enough to avoid abelian 4th powers [9]. Thirteen years later, Keränen proved that abelian squares are avoidable over four letters, settling the problem of classifying all the unary patterns in the abelian sense [11]: α is abelian unavoidable, $\alpha\alpha$ is abelian 4-avoidable but not abelian 3-avoidable, $\alpha\alpha\alpha$ is abelian 3-avoidable but not abelian 2-avoidable, and α^n, $n \geq 4$, is abelian 2-avoidable (where a pattern p is *k-avoidable* if there is an infinite abelian p-free word, i.e., not containing any occurrence of p, over a k-letter alphabet). For more results on abelian pattern avoidance, see [5–7].

In [3], the investigation of abelian avoidability in partial words was initiated. A partial word is *abelian square-free* if it does not contain any factor that results in an abelian square after filling in the holes. For example, $abc\diamond cb$ is an abelian square because we can replace the $\diamond$ by letter a and get $abcacb$. Blanchet-Sadri et al. constructed a partial word with infinitely many holes over five letters that is abelian square-free (except for trivial squares of the form $a\diamond$ or $\diamond a$, where a is a letter), and proved that none exists over four letters. In [4], Blanchet-Sadri et al. looked at the abelian avoidance of the patterns α^n in infinite partial words, where $n > 2$. They investigated, for a given n, the smallest alphabet size needed to construct an infinite partial word with finitely or infinitely many holes that avoids α^n in the abelian sense. They constructed in particular a binary partial word with infinitely many holes that avoids α^n, $n \geq 6$ in the abelian sense. Then, they proved that one cannot avoid α^n in the abelian sense under arbitrary insertion of holes in an infinite full word.

In this paper, we provide a more general study of abelian pattern avoidance in the context of partial words. We construct, given k-abelian avoidable patterns p in full words satisfying some conditions, abelian avoiding partial words with infinitely many holes over alphabet sizes that depend on k, as well as abelian avoiding infinite partial words with h holes over alphabet sizes that depend on k and h, for every integer $h > 0$. We construct, given infinite full words avoiding pattern p in the abelian sense, infinite abelian avoiding full words with the property that any of their positions can be changed to a hole while still avoiding p in the abelian sense. We also show that a pattern p with $n > 3$ distinct variables such that $|p| \geq 2^n$ is abelian avoidable by a partial word with infinitely many holes. The bound on $|p|$ turns out to be tight. We end up completing the classification of the binary and ternary patterns with respect to non-trivial abelian avoidability, in which no variable can be substituted by only one hole. We have put some sample proofs within the 12 page size limit to illustrate the different techniques we use. However the proofs of all our corollaries have been omitted due to the page constraint as well as our first proposition.

2 Preliminaries on Partial Words and Patterns

First, we recall some basic concepts of combinatorics on partial words; for more information, see [2]. A *partial word* over a finite alphabet A is a sequence of symbols from $A_\diamond = A \cup \{\diamond\}$, the alphabet A being augmented with the hole symbol $\diamond$; a *full word* is a partial word without holes. We denote by $u(i)$ the symbol at position i of a partial word u (labelling starts at 0). The *length* of u, denoted by $|u|$, represents the number of symbols in u. The *empty word* is the sequence of length zero and is denoted by ε. The set of all full words over A is denoted by A^*. Also, A^n denotes the set of all words over A of length n. A partial word u is a *factor* of a partial word v if there exist x, y such that $v = xuy$. We denote by $v[i..j]$ (resp., $v[i..j)$) the factor $v(i) \cdots v(j)$ (resp., $v(i) \cdots v(j-1)$). The powers of u are defined by $u^0 = \varepsilon$ and for $n \geq 1$, $u^n = uu^{n-1}$. If u, v are partial words of equal length, then u is *compatible* with v, denoted $u \uparrow v$, if $u(i) = v(i)$ for all i such that $u(i), v(i) \in A$.

Now, let us take a look at some concepts regarding patterns. Let E be a pattern alphabet and let $p = \alpha_0 \cdots \alpha_{n-1}$, where $\alpha_i \in E$. Define an *abelian occurrence* of p in a partial word w as a factor $u_0 \cdots u_{n-1}$ of w such that there exists a full word $v_0 \cdots v_{n-1}$, where for all i, $u_i \neq \varepsilon$ and $u_i \uparrow v_i$, and where for all i, j, if $\alpha_i = \alpha_j$, then v_i is a permutation of v_j.

The partial word w *meets* the pattern p in the abelian sense, or p *occurs* in w in the abelian sense, if for some factorization $w = xuy$, we have that u is an abelian occurrence of p in w; otherwise, w *avoids* p in the abelian sense (or w is *abelian p-free*). For instance, $ab\diamond ba\diamond baa$ meets $\alpha\beta\beta\alpha$ in the abelian sense (take $ab\underline{\diamond b}\ \underline{a}\ \underline{\diamond}\ \underline{ba}a$). An abelian occurrence of p is *trivial* if $u_i = \diamond$ for some i; otherwise, it is *non-trivial.* We call w *non-trivially abelian p-free* if it contains no non-trivial abelian occurrences of p. These definitions also apply to full words over A as well as infinite partial words over A which are functions from $\mathbb{N}$ to $A_\diamond$.

A pattern p is *k-abelian avoidable in partial words* if there is a word with infinitely many holes over an alphabet of size k that avoids p in the abelian sense; otherwise, p is *k-abelian unavoidable.* A pattern which is k-abelian avoidable (resp., k-abelian unavoidable) for some k (resp., every k) is called *abelian avoidable* (resp., *abelian unavoidable*). The *abelian avoidability index* of p is the smallest integer k such that p is k-abelian avoidable, or is ∞ if p is abelian unavoidable. Note that k-abelian avoidability implies $(k+1)$-abelian avoidability.

Proposition 1. *Let p be an abelian unavoidable pattern in full words. Then every word with infinitely many holes non-trivially meets p in the abelian sense.*

3 Avoiding Partial Words with Infinitely Many Holes

Our next goal is to find a bound so that if a pattern has length at least this bound, then it is abelian avoidable in partial words. We begin by considering one such bound. We will later improve upon it, but we need it in order to prove our tight bound, so we include it here.

Theorem 1. *Let p be a pattern with n distinct variables. If $|p| \geq 3 \times 2^{n-1}$, then there exists a partial word with infinitely many holes that is abelian p-free.*

Proof. We proceed by induction on n. If $n = 1$, then $p = \alpha^m$ for some $\alpha \in E, m \geq 3$. Assume the claim holds up to $n-1$ distinct variables. If there is some variable that occurs in p exactly once, p has a factor q with at most $n-1$ distinct variables with $|q| \geq 3 \times 2^{n-2}$, so by the inductive hypothesis a word with infinitely many holes exists that avoids q, and thus p, in the abelian sense. Therefore assume that every variable occurs at least twice. Then, by [5, Lemma 7], p can be avoided in the abelian sense by an infinite full word w over some alphabet A. There exist $a_0, a_1, a_2, a_3, a_4 \in A$ so that $a_0a_1a_2a_3a_4$ occurs infinitely often as a factor of w. Define $5 < k_0 < k_1 < \cdots$ so that $w[k_i-2..k_i+2] = a_0a_1a_2a_3a_4$ and $|p|k_i < k_{i+1}$, for all i. Consider $b_0, \ldots, b_4 \notin A$, and set $B = A \cup \{b_0, \ldots, b_4\}$. Define w' as follows:

$$w'(j) = \begin{cases} \diamond, & \text{if } j = k_i \quad\ \ \text{for some } i,\ i \equiv 0 \bmod 4|p|; \\ b_0, & \text{if } j = k_i - 2 \text{ for some } i,\ i \equiv 0 \bmod 4|p|; \\ b_1, & \text{if } j = k_i - 1 \text{ for some } i,\ i \equiv 0 \bmod 4|p|; \\ b_3, & \text{if } j = k_i + 1 \text{ for some } i,\ i \equiv 0 \bmod 4|p|; \\ b_4, & \text{if } j = k_i + 2 \text{ for some } i,\ i \equiv 0 \bmod 4|p|; \\ b_2, & \text{if } j = k_i \quad\ \ \text{for some } i,\ i \not\equiv 0 \bmod 4|p|; \\ w(j), & \text{otherwise.} \end{cases}$$

We claim that w' avoids p in the abelian sense. To see this, let $m = |p|$ and write $p = \alpha_0 \cdots \alpha_{m-1}$, where $\alpha_i \in E$. Assume that w' meets p in the abelian sense. Let $u_0 \cdots u_{m-1}$ be an occurrence, where $u_j = w'[i_j..i_{j+1})$. Each u_i contains at most one b_2. To see this, assume there exists $u_{j'}$ containing two b_2's. So there is some s such that $i_{j'} \leq k_s < k_{s+1} < i_{j'+1}$. Let u_j be chosen so that $j \neq j'$ and $\alpha_j = \alpha_{j'}$ (we can do this since each variable that occurs in p occurs at least twice). Then for each b_2 that $u_{j'}$ contains, u_j contains either $\diamond$ or b_2. So there is some t satisfying $i_j \leq k_t < k_{t+1} < i_{j+1}$. Assume that $j' < j$, the other case being similar. We have $|u_{j'}| \leq i_{j'+1} \leq i_j \leq k_t < k_{t+1} - k_t < i_{j+1} - i_j = |u_j| = |u_{j'}|$, which is a contradiction, proving the claim.

Note that $u_0 \cdots u_{m-1}$ contains a hole, since otherwise we can replace each b_i with an a_i to get $u'_0 \cdots u'_{m-1}$, which is still an abelian occurrence of p and a factor of w, yielding a contradiction. Therefore u_j contains a hole, for some j. Also, there exists j_0 so that u_{j_0} contains one of the letters b_0, b_1, b_3, b_4, and so that there is some j_1 distinct from both j_0 and j satisfying $\alpha_{j_0} = \alpha_{j_1}$ (where j is as above). To see this, since each $\diamond$ occurs in a factor of the form $b_0 b_1 \diamond b_3 b_4$, the claim follows trivially if $u_j \neq \diamond$ (by letting $j_0 = j$, each variable occurring more than once in p). Therefore suppose that $u_j = \diamond$. Let $j > 1$, the cases $j = 1$ and $j = 0$ being similar. We have $j - 2 \geq 0$, so u_{j-1} and u_{j-2} are well defined. Note that u_{j-1} contains b_1. If $\alpha_{j-1} \neq \alpha_j$, since each variable occurs in p at least twice, there is some j_1 distinct from both $j - 1$ and j so that $\alpha_{j_1} = \alpha_{j-1}$, and set $j_0 = j - 1$. If $\alpha_{j-1} = \alpha_j$ then $|u_{j-1}| = |u_j| = 1$, so $u_{j-1} = b_1$. Note that u_{j-2} contains b_0, and that $\alpha_{j-2} \neq \alpha_{j-1} = \alpha_j$. Since each variable in p occurs at least twice, there exists j_1 distinct from $j - 2$ so that $\alpha_{j-2} = \alpha_{j_1}$, and set $j_0 = j - 2$.

Since u_{j_0} contains b_i, for some $i \in \{0, 1, 3, 4\}$, u_{j_1} contains b_i or $\diamond$. So there exist s_0 and s_1 such that $s_0 \equiv s_1 \equiv 0 \bmod 4|p|$, $i_{j_0} \leq k_{s_0} + i - 2 < i_{j_0+1}$, and $i_{j_1} \leq k_{s_1} + i - 2 < i_{j_1+1}$ or $i_{j_1} \leq k_{s_1} < i_{j_1+1}$. However, since $s_0 \equiv s_1 \equiv 0 \bmod 4m$ and $s_0 \neq s_1$, there are at least $4m - 1$ integers between s_0 and s_1. For each such integer s, we have $w'(k_s) = b_2$. This implies that $u_0 \cdots u_{m-1}$ contains at least $4m - 1$ b_2's. Then by the pigeonhole principle, there is at least one u_l containing three b_2's. However, this is impossible. □

By considering the pattern $p = \alpha\alpha$, we can see that the bound in Theorem 1 is tight over a unary alphabet of variables. Moreover, it is tight over a binary alphabet of variables as implied by the following.

Corollary 1. *Let p be an avoidable pattern in full words over the alphabet $\{\alpha, \beta\}$. Then p is avoided by a partial word with infinitely many holes in the abelian sense if and only if p is not a subpattern of either $\alpha\alpha\beta\alpha\alpha$ or $\beta\beta\alpha\beta\beta$.*

Our next step is to strengthen Theorem 1 for pattern alphabets of at least three variables. For this, we need the following proposition.

Proposition 2. *Let p be a pattern over an alphabet E such that $p \neq \alpha\alpha$, for any $\alpha \in E$. If each variable in p occurs at least twice, then p can be avoided in the abelian sense by a partial word with infinitely many holes. In particular, if p is abelian k-avoidable, then there exists a partial word with infinitely many holes over an alphabet of size $k + 4$ that is abelian p-free.*

Proof. Since each variable in $p = \alpha_0 \cdots \alpha_{m-1}$ occurs at least twice, by [5], p can be avoided in the abelian sense by an infinite full word w over some alphabet A of size k. Let $B = A \cup \{a, b, c, d\}$ where $a, b, c, d \notin A$. Since there is some factor v, $|v| = 5$, that occurs infinitely often in w, consider a sequence $5 < k_0 < k_1 < k_2 < \cdots$ so that $w[k_i - 2..k_i + 2] = v$ and $2k_i < k_{i+1}$, for all i. Then define w' by

$$w'(j) = \begin{cases} a, & \text{if } j = k_i - 2 \text{ for some } i,\ i \equiv 0 \bmod 4|p|; \\ b, & \text{if } j = k_i - 1 \text{ for some } i,\ i \equiv 0 \bmod 4|p|; \\ \diamond, & \text{if } j = k_i \quad\ \ \text{for some } i,\ i \equiv 0 \bmod 4|p|; \\ c, & \text{if } j = k_i + 1 \text{ for some } i,\ i \equiv 0 \bmod 4|p|; \\ a, & \text{if } j = k_i + 2 \text{ for some } i,\ i \equiv 0 \bmod 4|p|; \\ d, & \text{if } j = k_i \quad\ \ \text{for some } i,\ i \not\equiv 0 \bmod 4|p|; \\ w(j), & \text{otherwise.} \end{cases}$$

For the sake of contradiction, suppose that w' contains $u = u_0 u_1 \cdots u_{m-1}$ as an abelian occurrence of p and write $u_j = w'[i_j..i_{j+1})$. Note that $u_0 u_1 \cdots u_{m-1}$ contains a hole since otherwise w would have a factor that is an abelian occurrence of p. Moreover, u contains at least $4|p| - 1$ d's. To see this, assume that u_j contains a hole. Then there are two cases to consider. If $u_j \neq \diamond$, then u_j contains b or c, since every $\diamond$ in w' occurs in a factor of the form $ab\diamond ca$. Without loss of generality, assume u_j has $b\diamond$ as a factor. Since every variable in p occurs at least twice, there is some j', distinct from j, so that $u_{j'}$ is compatible with a permutation of u_j. Thus $u_{j'}$ contains either b or $\diamond$. Suppose $j' > j$, the case $j' < j$ being similar. Then there exists some s such that $i_j \leq k_s < i_{j+1}$, and some t such that $i_{j'} < k_t \leq i_{j'+1}$ or $i_{j'} \leq k_t < i_{j'+1}$, where $s < t$, $s \equiv t \equiv 0 \bmod 4|p|$. Thus $i_0 \leq k_s < k_{s+4|p|} \leq k_t \leq i_m$. So there are $4|p| - 1$ l's so that $k_s < k_l < k_{s+4|p|}$. Since for each $l \not\equiv 0 \bmod 4|p|$, $w'(k_l) = d$, there are $4|p| - 1$ d's in u. If $u_j = \diamond$, then a similar reasoning works.

Since u contains at least $4|p| - 1 = 4m - 1$ d's, it follows from the pigeonhole principle that some u_j contains at least two d's. So there is an s such that $i_j \leq k_s < k_{s+1} < i_{j+1}$. Since there is a j' such that $u_{j'}$ is compatible with a permutation of u_j, let $j' > j$ (the other case is similar). Here, $u_{j'}$ contains either a d or a $\diamond$ for each d in u_j. This implies there is some t such that $i_{j'} \leq k_t < k_{t+1} < i_{j'+1}$, where $t > s$. Then

$$|u_j| \leq i_{j+1} \leq i_{j'} \leq k_t < k_{t+1} - k_t < i_{j'+1} - i_{j'} = |u_{j'}| = |u_j|.$$

This is a contradiction, proving the claim. □

Theorem 2. *Let p be a pattern over an alphabet E. Then there exists a partial word with infinitely many holes that is abelian p-free if one of the following holds: (1) $\|E\| = 3$ and $|p| \geq 9$; (2) $\|E\| > 3$ and $|p| \geq 2^{\|E\|}$.*

Proof. The proof of Statement 2 is omitted due to the 12 page constraint. For Statement 1, let $p = \alpha_0 \cdots \alpha_{m-1} \in E^m$ be a pattern over the alphabet $E = \{\alpha, \beta, \gamma\}$. It is sufficient to consider the case where $|p| = 9$. For the sake of a contradiction, suppose that there is no partial word with infinitely many holes that avoids p in the abelian sense. At least one variable in p occurs exactly once (the case when each variable in p occurs at least twice follows from Proposition 2). Without loss of generality, we can assume that this variable is γ. Therefore we can write $q_0\gamma q_1 = p$, where q_0 and q_1 are patterns over $\{\alpha, \beta\}$. There exists no partial word with infinitely many holes that avoids q_0 or q_1 in the abelian sense, because that word should also avoid p in the abelian sense. By Theorem 1, $|q_i| < 6$ for both i. Since $|q_0| + |q_1| = 9 - 1 = 8$, $(|q_0|, |q_1|) \in \{(5, 3), (3, 5), (4, 4)\}$.

There exists an infinite word w over an alphabet A that avoids abelian squares. Assume that $a_0a_1a_2 \in A^3$ occurs infinitely often in w, then consider any sequence $4 < k_0 < k_1 < \cdots$ where $k_{i+1} > 3k_i$ and $w[k_i - 1..k_i + 1] = a_0a_1a_2$, for all i. Moreover, consider $a, b, c \notin A$, where a, b and c are distinct letters. Let $B = A \cup \{a, b, c\}$. We can then define a partial word w' over B as follows:

$$w'(j) = \begin{cases} a, & \text{if } j = k_i - 1 \text{ for some } i,\ i \equiv 0 \bmod 3; \\ \diamond, & \text{if } j = k_i \quad\ \ \text{ for some } i,\ i \equiv 0 \bmod 3; \\ b, & \text{if } j = k_i + 1 \text{ for some } i,\ i \equiv 0 \bmod 3; \\ c, & \text{if } j = k_i \quad\ \ \text{ for some } i,\ i \not\equiv 0 \bmod 3; \\ w(j), & \text{otherwise.} \end{cases}$$

The partial word w' avoids non-trivial abelian squares, following the same argument as in the proof of Theorem 2 in [3]. We want to show that w' avoids p in the abelian sense. For the sake of a contradiction, suppose that $u_0 \cdots u_8$ is an abelian occurrence of p. Set $u_j = w'[i_j..i_{j+1})$ for each j.

First, consider the case $|q_0| = 3$, $|q_1| = 5$. Since $|q_1| > 4$ we know that q_1 is avoidable in full words. By Corollary 1, $q_1 = \alpha\alpha\beta\alpha\alpha$ or $q_1 = \beta\beta\alpha\beta\beta$. Without loss of generality we assume that $q_1 = \alpha\alpha\beta\alpha\alpha$. Then $u_4u_5u_6u_7u_8$ is an abelian occurrence of q_1. Since u_4u_5 and u_7u_8 are both abelian squares and w' avoids non-trivial abelian squares, we get that u_4u_5 and u_7u_8 both contain a hole. Thus there is some s such that $i_4 \leq k_s < k_{s+1} < i_9$. Moreover, β occurs in q_0, since otherwise $q_0 = \alpha\alpha\alpha$. If u_i corresponds to some β in q_0, we get that

$$|u_i| = |u_6| = |u_6| + 4 - 4 = |u_4u_5u_6u_7u_8| - 4 \geq k_{s+1} - k_s - 4 > k_s > |u_i|.$$

This, however, is a contradiction.

Next, consider the case where $|q_0| \geq 4$. This implies that q_0 is a subpattern of $\alpha\alpha\beta\alpha\alpha$ or $\beta\beta\alpha\beta\beta$. Without loss of generality we can assume it is a subpattern of $\alpha\alpha\beta\alpha\alpha$. Moreover, since $|q_0| \geq 4$, q_0 contains $\alpha\alpha$ and $\alpha\beta\alpha$. Therefore there exists some j so that $\alpha_j = \alpha_{j+1} = \alpha$. Thus u_ju_{j+1} is an abelian square. Since w' avoids non-trivial abelian squares, u_ju_{j+1} must be trivial. Thus u_ju_{j+1} is a factor of $a\diamond b$ of length two. Therefore we either have that $u_j = a$ and $u_{j+1} = \diamond$ or $u_j = \diamond$ and $u_{j+1} = b$. Consider the first case, the other being similar. There is l, $l < 4$, so that $\alpha_l\alpha_{l+1}\alpha_{l+2} = \alpha\beta\alpha$. Since $u_l, u_{l+2} \in \{a, \diamond\}$, there exist s_0 and s_1 so that either $i_l = k_{s_0} - 1$ or $i_l = k_{s_0}$, and either $i_{l+2} = k_{s_1} - 1$ or $i_{l+2} = k_{s_1}$, because $i_l = i_{l+1} - 1$, $i_{l+2} = i_{l+3} - 1$ due to the fact that $|u_l| = |u_{l+2}| = 1$. However, note that if $s_0 = s_1$ we get that $k_{s_0} - 1 \leq i_l < i_{l+1} \leq i_{l+2} - 1 \leq k_{s_1} - 1 = k_{s_0} - 1$, which is a contradiction. Thus we get that $i_l \leq k_{s_0} < k_{s_1} \leq i_{l+2}$. As above this leads to a contradiction. □

The bounds in Theorem 2 are tight. For (1), $\alpha\alpha\beta\alpha\alpha\gamma\alpha\alpha$ cannot be avoided by any partial word with infinitely many holes (since there must be some $a \in A_\diamond$ such that $a\diamond$ occurs infinitely often). For (2), the nth Zimin pattern, Z_n, is abelian unavoidable and is of length $2^n - 1$ (see Chapter 3 of [13]). More specifically, if $E = \{\alpha_0, \ldots, \alpha_{n-1}\}$, then $Z_m = \alpha_0$ if $m = 1$, and $Z_m = Z_{m-1}\alpha_{m-1}Z_{m-1}$ if $1 < m \leq n$. Meanwhile, we can consider the case of non-trivial avoidance.

Theorem 3. *Let p be a pattern with each of its variables occurring at least twice that is abelian k-avoidable in full words. Then a partial word with infinitely many holes over an alphabet of size $k+2$ exists that is non-trivially abelian p-free.*

Proof. Suppose w is a word over an alphabet A of size k that avoids p in the abelian sense. Then we define a sequence of k_i's, where $k_{i+1} > 5k_i$. Let $B = A \cup \{c, d\}$, where $c, d \notin A$. Then a partial word w' is defined as follows: $w'(i) = \diamond$ if $i = k_j$ for some j, $j \equiv 0 \bmod 6|p|$, $w'(i) = c$ if $i = k_j - 1$ or $i = k_j + 1$ for some j, $j \equiv 0 \bmod 6|p|$, and $w'(i) = d$ if $i = k_j$ for some j, $j \not\equiv 0 \bmod 6|p|$, and $w'(i) = w(i)$ otherwise. Then w' contains infinitely many holes. Set $p = \alpha_0 \cdots \alpha_{n-1} \in E^n$, and assume that $u = u_0 \cdots u_{n-1} = w'[j_0..j_1]$ is a non-trivial abelian occurrence of p in w', i.e., if $\alpha_i = \alpha_j$ then u_i is compatible with a permutation of u_j, and $u_i \neq \diamond$, for $0 \leq i < n$.

Let α_i and $\alpha_{i'}$ be two occurrences of any variable α in p, where $i < i'$, and suppose u_i and $u_{i'}$ are the corresponding partial words. Then there exist s, t such that $i \leq s < s + l < t < t + l \leq i'$, where $u_i = w'[s..s + l]$ and $u_{i'} = w'[t..t + l]$. Let $J_1 = \{j \mid s \leq k_j \leq s + l\}$ and $J_2 = \{j \mid t \leq k_j \leq t + l\}$. Then $\|J_1\| < 3$ and $\|J_2\| < 2$. To see this, if $\|J_2\| \geq 2$ then there exist $j, j+1 \in J_2$, and $l = t + l - t \geq k_{j+1} - k_j > k_j > s + l \geq l$ is a contradiction. If $\|J_1\| \geq 3$ then there are at least 2 d's in u_i, which means that $u_{i'}$ has to contain at least 2 d's or $\diamond$'s. Then J_2 contains at least 2 j's, which is a contradiction. Therefore $\|J_1\| < 3$ and $\|J_2\| < 2$. Since for each α in p, $\|J_1 \cup J_2\| \leq 3$, there are at most $3|p|$ integer j's such that $j_0 \leq k_j \leq j_1$.

Now we show that no u_i contains a hole. Suppose some u_i contains one. Then c must also occur in u_i since $|u_i| > 1$. Then either $\diamond c$ or $c\diamond$ occurs in $u_{i'}$, which is compatible with a permutation of u_i. Suppose $\diamond c$ occurs in $u_{i'}$, the other case is similar. Then between the occurrence of $\diamond$ in u_i and $\diamond$ in $u_{i'}$ there are at least $6|p| - 2$ k_j's. This contradicts the fact that there are at most $3|p|$ k_j's from position j_0 to position j_1. Therefore u contains no holes. Then we can replace all the occurrences of c by a_1 and d by a_2, for some $a_1, a_2 \in A$, and we get an abelian occurrence of p. However, this contradicts the fact that p can be avoided in the abelian sense by infinite full words over k letters. Therefore w' non-trivially avoids p in the abelian sense. □

Corollary 2. *Let p be a pattern with n distinct variables. If $|p| \geq 2^n$, then there exists a finite alphabet A and a partial word with infinitely many holes over A that is non-trivially abelian p-free.*

Corollary 3. *If p is an abelian avoidable binary pattern in full words, then a partial word with infinitely many holes exists that is non-trivially abelian p-free.*

We also give the following bounds.

Proposition 3. *Let p be a binary pattern. If $|p| \geq 360$ (resp., $|p| \geq 118$), then a partial word over a binary (resp., ternary) alphabet with infinitely many holes exists that is abelian p-free.*

Proof. Let $p = \alpha_0 \cdots \alpha_{n-1}$ be a pattern over $E = \{\alpha, \beta\}$ such that $|p| \geq 360$. Without loss of generality we can assume that $\alpha_0 = \alpha$. Note that $\alpha_i = \beta$ for some $i < 6$, since otherwise p contains α^6 as a subpattern, which can be abelian avoided by a binary partial word with infinitely many holes [4]. Then write $p = qq_0q_1q_2$ where $|q_i| = 118$ for each i, and $|q| \geq 6$. We can do this since $|p| \geq 360 = 3 \times 118 + 6$. From [7, Lemma 3.2], there exist infinite binary words w_2 and w_3 so that every binary pattern of length at least 118 is avoided in the abelian sense by either w_2 or w_3. By the pigeonhole principle there exist $l < l'$, where q_l and $q_{l'}$ are either both avoided in the abelian sense by w_2 or both avoided in the abelian sense by w_3. Assume that $l = 0$ and $l' = 1$, the other cases being similar. Let w be the infinite binary word that avoids q_0 and q_1 in the abelian sense. Define a sequence $3 < k_0 < k_1 < k_2 < \cdots$ so that $k_{i+1} > (|p| + 1)k_i$ for all i. We can then define the partial word w' by $w'(j) = \diamond$ if $j = k_i$ for some i, and $w'(j) = w(j)$ otherwise.

For the sake of a contradiction, suppose that $u = u_0 \cdots u_{n-1}$ is an abelian occurrence of p in w', and let $v = u_6 \cdots u_{n-1}$. Set $u_j = [i_j..i_{j+1})$ for each j. Then there is at most one hole in v. To see this, assume there are two, and write $v = w'[i_6..i_n)$. Then there is an s so that $i_6 \leq k_s < k_{s+1} < i_n$. Since $u_0 \cdots u_5$ occurs in $w'[0..i_6)$ this implies that $|u_j| \leq i_6$ for all j. Thus $i_6 + |u| = i_6 + |u_0| + \cdots + |u_{n-1}| \leq (n + 1)i_6 \leq (n + 1)k_s < k_{s+1} < i_n = i_6 + i_n - i_6 = i_6 + |v| \leq i_6 + |u|$, yielding a contradiction, so v contains at most one hole. Then $q_0 = \alpha_{|q|} \cdots \alpha_{|q|+117}$ and $q_1 = \alpha_{|q|+118} \cdots \alpha_{|q|+235}$. Note that $u_{|q|} \cdots u_{|q|+117}$ and $u_{|q|+118} \cdots u_{|q|+235}$ are both factors of v, then at most one of them contains a hole. Without loss of generality we assume that $u_{|q|} \cdots u_{|q|+117}$ does not contain any holes. However, $u_{|q|} \cdots u_{|q|+117}$ is an abelian occurrence of q_0 in w' that is a full word, so it must be a factor of w. This contradicts the fact that w avoids q_0 in the abelian sense.

Now, let $p = \alpha_0 \cdots \alpha_{n-1}$ be a pattern over $E = \{\alpha, \beta\}$ such that $|p| \geq 118$. As above, we can assume that $\alpha_0 = \alpha$. Note that $\alpha_i = \beta$ for some $i < 6$, since otherwise p contains α^6 as a subpattern, which can be avoided by a binary partial word with infinitely many holes in the abelian sense. From [7, Lemma 3.2], there exists an infinite word w over $A = \{a, b\}$ that avoids abelian occurrences of p. Moreover, bab occurs infinitely often as a factor of w (this can be seen by considering the words w_2 and w_3 in [7]). Define a sequence $3 < k_0 < k_1 < k_2 < \cdots$ so that $(|p|+1)k_i < k_{i+1}$ and $w[k_i - 1..k_i + 1] = bab$, for all i. Consider $c \notin A$. We can then define a partial word w' as follows: $w'(j) = \diamond$ if $j = k_i$ for some i, $w'(j) = c$ if $j = k_i - 1$ or $j = k_i + 1$ for some i, and $w'(j) = w(j)$ otherwise.

For the sake of a contradiction, suppose that $u_0 \cdots u_{n-1}$ is an abelian occurrence of p in w'. Since w avoids abelian occurrences of p, some u_j must contain a hole. However, since each $\diamond$ in w' occurs in a factor of the form $c\diamond c$, some u_i must contain c. Assume that $\alpha_i = \beta$, the other case being similar. Using an argument similar to that above, we can argue that there is at most one j so that $j \geq 6$ and so that u_j contains a hole. Then one variable in $\alpha_7 \cdots \alpha_{13}$ must be β. Let α_l be this variable. Then u_l contains either a c or a $\diamond$. Since each c occurs next to a $\diamond$, $u_{l-1}u_lu_{l+1}$ contains a hole, where $l-1 \geq 6$ and $l+1 \leq 14$. Similarly, there exists an l' so that $23 \geq l' \geq 17$, and where $u_{l'-1}u_{l'}u_{l'+1}$ contains a hole. However, this contradicts the fact that there exists at most one j, $j \geq 6$, such that u_j contains a hole. Thus w' avoids p in the abelian sense. □

4 Avoiding Partial Words with Finitely Many Holes

Now, we give constructions for avoiding words with finitely many holes.

Theorem 4. *If p is an abelian k-avoidable pattern, then for every integer $h \geq 0$ there exists an infinite word with h holes over an alphabet of size $k+2h$ that is non-trivially abelian p-free.*

Proof. Since p is abelian avoidable, $|p| \geq 2$ and we write $p = \alpha\beta q$, where q is a word and α, β are variables. If $\alpha = \beta$, p contains a square, and from [3], a word with infinitely many holes over a five-letter alphabet can be constructed that avoids abelian squares. If we only put h holes instead of infinitely many in that construction, p should still be abelian avoidable. Therefore we only need to consider when $\alpha \neq \beta$.

First consider the case where α and β are both contained in q. Since p is abelian k-avoidable, there exists an infinite word w over a k-letter alphabet A such that w avoids p in the abelian sense. Let $C = \{a_0, \ldots, a_{h-1}\} \cup \{b_0, \ldots, b_{h-1}\}$, where $A \cap C = \emptyset$. Define $A' = A \cup C$, so $\|A'\| = k + 2h$. Then an infinite partial word w' over A' is defined as follows:

$$w'(i) = \begin{cases} a_j, & \text{if } i = 4j \text{ for some } j,\ 0 \leq j < h; \\ \diamond & \text{if } i = 4j+1 \text{ for some } j,\ 0 \leq j < h; \\ b_j, & \text{if } i = 4j+2 \text{ for some } j,\ 0 \leq j < h; \\ a_j, & \text{if } i = 4j+3 \text{ for some } j,\ 0 \leq j < h; \\ w(i), & \text{otherwise.} \end{cases}$$

Then w' is of the form $a_0\diamond b_0a_0a_1\diamond b_1a_1 \cdots a_{h-1}\diamond b_{h-1}a_{h-1}w(4h)w(4h+1)\cdots$. Let $p = \alpha_0 \cdots \alpha_{m-1}$, $\alpha_i \in E$. Note that w' has h holes and each a_j and b_j only appears once. Suppose there exists $u = u_0u_1 \cdots u_{m-1}$ that is a non-trivial abelian occurrence of p in w'. Then at least u_0 contains a hole. Since u is a non-trivial occurrence, $|u_0| \geq 2$. Then there exists $l > 1$ such that u_l is compatible with a permutation of u_0. If $|u_0| \geq 3$, then $u_0 = a_i\diamond b_i$ and $u_l = a_ia_{i+1}\diamond$, which means u_0 and u_l are consecutive; however, this implies $u_l = u_1$, but u_1 is not compatible with a permutation of u_0. Thus u_l contains a hole and u_0 must be of the form $a\diamond$ or $\diamond b$ to be compatible with u_l in the abelian sense, where $a \in \{a_0, \ldots, a_{h-1}\}$ and $b \in \{b_0, \ldots, b_{h-1}\}$. Moreover, since $l > 1$, $u_1 \in C_\diamond^*$ as well. Since β appears again in q, using the same logic as above, u_1 must also be of the form $a\diamond$ or $\diamond b$ for some a, b. However, such u_0u_1 cannot be a factor of w'.

We prove the rest of the claim by induction on the length of p. The case of $|p| = 1$ is trivial. When α is not contained in q, if $\alpha\beta q$ is abelian k-avoidable, so is βq. Since $|\beta q| < |\alpha\beta q|$, there exists an infinite partial word with h holes over a $(k+2h)$-letter alphabet that non-trivially avoids βq in the abelian sense, thus non-trivially avoiding $\alpha\beta q$ in the abelian sense. Similarly, if β does not occur in q, then q is non-trivially abelian avoidable over an alphabet of size $k+2h$, and so are βq and $\alpha\beta q$, which is p. □

Corollary 4. *Let p be an abelian k-avoidable pattern in full words with each of its variables occurring at least twice. Then for every integer $h \geq 0$ there exists an infinite word with h holes over an alphabet of size $k+2h$ that is abelian p-free.*

The following result is concerned with an arbitrary insertion of a hole in an infinite word that avoids a pattern in the abelian sense.

Theorem 5. *If p is an abelian avoidable pattern, then there exists an infinite abelian avoiding full word so that we can insert a hole into any position and get a partial word that is non-trivially abelian p-free.*

Proof. Assume $p = \alpha_0 \cdots \alpha_{n-1}$ is abelian avoided by an infinite word w over alphabet A. Let $A' = A^2 \times \{0,1\}$, and define an infinite word v as follows. Let $j \in \{0,1\}$. For integer $i \geq 0$, if $i \equiv j \bmod 2$, let $v(i) = (w(i-j)w(i-j+1), j) \in A'$. Suppose towards a contradiction that v is not abelian p-free. Thus, there exist i_1 and i_2 such that $\alpha_{i_1} = \alpha_{i_2}$, $w[i_1..i_1+l]$ and $w[i_2..i_2+l]$ are not permutations of each other, while $v[i_1..i_1+l]$ and $v[i_2..i_2+l]$ are. This means $w(i_1-j)w(i_1-j+1)\cdots w(i_1+l-j)w(i_1+l-j+1)$ is a permutation of $w(i_2-j)w(i_2-j+1)\cdots w(i_2+l-j)w(i_2+l-j+1)$. Since $j = 0$ or 1, this contradicts the fact that w avoids p in the abelian sense.

Assume we replace one letter in v with a $\diamond$ to get v', and $u = u_0u_1\cdots u_{n-1}$ is a non-trivial abelian occurrence of p in v'. Note that u must contain the hole. Suppose u_s contains the hole and u_s is compatible with a permutation of u_t, for some s, t. Then since $|u_s| > 1$, either $(w(i_1-j_1)w(i_1-j_1+1), j_1)\diamond$ or $\diamond(w(i_1-j_1)w(i_1-j_1+1), j_1)$ is a factor of u_s, for some integers i_1, j_1. This means u_t contains a letter $(w(i_2-j_2)w(i_2-j_2+1), j_2)$ where $j_2 = j_1$, $w(i_2-j_2) = w(i_1-j_1)$, and $w(i_2-j_2+1) = w(i_1-j_1+1)$. Since j_1, j_2 is either 0 or 1, $w(i_1) = w(i_2)$ for either case. This contradicts w avoiding p in the abelian sense. Thus v is abelian p-free even after an arbitrary insertion of a hole. □

5 Concluding Remarks

By Corollary 2, the ternary patterns of length ≥ 8 are non-trivially abelian avoidable by partial words with infinitely many holes. Moreover, a word with infinitely many holes over five letters exists that avoids non-trivial abelian squares [3]. Therefore, patterns containing abelian squares are non-trivially abelian avoidable as well, and we only need to examine patterns of length at most 7 without any abelian squares. Currie et al. in [5] characterize these remaining six patterns: $\alpha\beta\alpha\gamma\alpha\beta\alpha$, $\alpha\beta\gamma\alpha\beta\alpha$ and $\alpha\beta\gamma\alpha\gamma$ are abelian unavoidable, while $\alpha\beta\alpha\gamma\beta\alpha\beta$,

$\alpha\beta\alpha\gamma\beta\gamma$ and $\alpha\beta\gamma\beta\alpha\beta\gamma$ are abelian avoidable. By Proposition 1, there is no word with arbitrarily many holes that non-trivially avoids $\alpha\beta\alpha\gamma\alpha\beta\alpha$, $\alpha\beta\gamma\alpha\beta\alpha$ or $\alpha\beta\gamma\alpha\gamma$ in the abelian sense. By Theorem 3, since each variable occurs at least twice in $\alpha\beta\alpha\gamma\beta\gamma$ and $\alpha\beta\gamma\beta\alpha\beta\gamma$, partial words with infinitely many holes exist that non-trivially avoid them in the abelian sense. The last pattern, $\alpha\beta\alpha\gamma\beta\alpha\beta$, can be shown to be abelian avoidable by a word with infinitely many holes with a proof similar to that of Lemma 8 in [5]. Therefore, the non-trivial abelian avoidability of the ternary patterns in the context of partial words is complete.

By Corollary 3, if a binary pattern is abelian avoidable by a full word, then it is also non-trivially abelian avoidable by a partial word with infinitely many holes. Combining this with Proposition 1, the non-trivial abelian avoidability of binary patterns in partial words is the same as in full words.

References

1. Berstel, J., Boasson, L.: Partial words and a theorem of Fine and Wilf. Theoretical Computer Science 218, 135–141 (1999)
2. Blanchet-Sadri, F.: Algorithmic Combinatorics on Partial Words. Chapman & Hall/CRC Press, Boca Raton, FL (2008)
3. Blanchet-Sadri, F., Kim, J.I., Mercaş, R., Severa, W., Simmons, S., Xu, D.: Avoiding abelian squares in partial words. Journal of Combinatorial Theory, Series A 119, 257–270 (2012)
4. Blanchet-Sadri, F., Simmons, S., Xu, D.: Abelian repetitions in partial words. Advances in Applied Mathematics 48, 194–214 (2012)
5. Currie, J., Linek, V.: Avoiding patterns in the abelian sense. Canadian Journal of Mathematics 53, 696–714 (2001)
6. Currie, J., Visentin, T.: On abelian 2-avoidable binary patterns. Acta Informatica 43, 521–533 (2007)
7. Currie, J., Visentin, T.: Long binary patterns are abelian 2-avoidable. Theoretical Computer Science 409, 432–437 (2008)
8. Currie, J.D.: Pattern avoidance: themes and variations. Theoretical Computer Science 339, 7–18 (2005)
9. Dekking, F.M.: Strongly non-repetitive sequences and progression-free sets. Journal of Combinatorial Theory, Series A 27(2), 181–185 (1979)
10. Erdős, P.: Some unsolved problems. Magyar Tudományos Akadémia Matematikai Kutató Intézete Közl. 6, 221–254 (1961)
11. Keränen, V.: Abelian Squares are Avoidable on 4 Letters. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 41–52. Springer, Heidelberg (1992)
12. Leupold, P.: Partial Words for DNA Coding. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 224–234. Springer, Heidelberg (2005)
13. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)
14. Thue, A.: Über unendliche Zeichenreihen. Norske Vid. Selsk. Skr. I, Mat. Nat. Kl. Christiana 7, 1–22 (1906)

The Complexity of Rerouting Shortest Paths

Paul Bonsma

Computer Science Department, Humboldt University Berlin, Germany
bonsma@informatik.hu-berlin.de

Abstract. The Shortest Path Reconfiguration problem has as input a graph G (with unit edge lengths) with vertices s and t, and two shortest st-paths P and Q. The question is whether there exists a sequence of shortest st-paths that starts with P and ends with Q, such that subsequent paths differ in only one vertex. This is called a rerouting sequence.

This problem is shown to be PSPACE-complete. For claw-free graphs and chordal graphs, it is shown that the problem can be solved in polynomial time, and that shortest rerouting sequences have linear length. For these classes, it is also shown that deciding whether a rerouting sequence exists between *all* pairs of shortest st-paths can be done in polynomial time.

1 Introduction

In this paper, we study the *Shortest Path Reconfiguration (SPR) Problem*, introduced by Kamiński et al [15, 16]. The input consists of a graph G, with vertices s and t, and two shortest st-paths P and Q. The question is whether P can be modified to Q by changing one vertex at a time, and maintaining a shortest st-path throughout. Edges have unit lengths, so all shortest st-paths have the same number of vertices. We define the following *solution graph* $\mathrm{SP}(G, s, t)$: its vertex set is the set of all shortest st-paths in G. Two paths P and Q are adjacent if they differ in one vertex. SPR can now be reformulated as: does there exist a walk from P to Q in $\mathrm{SP}(G, s, t)$? Such a walk is also called a *rerouting sequence*.

Shortest paths form a central concept in graph theory, optimization, algorithms and networking. Questions related to rerouting (shortest) paths are often studied in networking applications. Although we are not aware of an application where this reachability question is studied, it is a very natural question, which may provide insight to practical rerouting problems. Nevertheless, the main motivation for this study is of a more theoretical nature. Similar reconfiguration problems can be defined based on many different combinatorial problems: Consider all solutions to a problem (or all solutions of at least/at most given weight, in the case of optimization problems), and define a (symmetric) adjacency relation on them. Such problems have been studied often in recent literature. Examples include reconfiguration problems based on satisfiability problems [10], independent sets [11, 13, 17], vertex colorings [1, 3–6], matchings [13], list edge-colorings [14], matroid bases [13], subsets of a (multi)set of numbers [9]. To obtain a reconfiguration problem, one needs to define an adjacency relation between solutions. Usually, the most natural adjacency relation is considered, e.g.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 222–233, 2012.

two independent sets I and J are considered adjacent in [13] if J can be obtained from I by removing one vertex and adding another; boolean assignments are considered adjacent in [10] if exactly one variable differs, etc. We remark that in the context of local search, similar problems have been studied earlier, with the important distinction that the neighborhood is not symmetric, and the objective is to reach a local optimum, instead of a given target solution, see e.g. [18].

An initial motivation of these questions was to explore the solution space of NP-hard problems, to study e.g. the performance of heuristics [10] and random sampling methods [4]. This has revealed interesting, often recurring patterns in the complexity behavior of these problems. This is perhaps best exemplified by the known results on the reconfiguration of vertex colorings using k colors: in the problem k-Color Path, two k-colorings of a graph are given, and the question is whether one can be modified to the other by changing one vertex color at a time, and maintaining a k-coloring throughout. This problem is polynomial time solvable for $k \leq 3$ [6], and PSPACE-complete for $k \geq 4$ [3]. Note that the corresponding decision problem of deciding whether a graph admits a k-coloring is polynomial time solvable for $k \leq 2$, and NP-complete for $k \geq 3$. This gives an example of the following common pattern: for instance classes for which deciding whether a solution exists is in P, the reconfiguration problem is often in P as well. See [10, 12, 13] for more extensive examples. This motivated Ito et al [12] to ask for examples of reconfiguration problems that break this pattern. Secondly, it has been observed that there is a strong correlation between the complexity of reconfiguration problems and the diameter of the components of the solution graph: for all known 'natural' reconfiguration problems in P, the diameter is polynomially bounded (see e.g. [1, 6, 10, 13, 17]), and for all PSPACE-complete reconfiguration problems, the diameter may be superpolynomial or exponential (see e.g. [3, 10]). The latter is unsurprising, since polynomial diameter would imply NP=PSPACE (assuming that the property of being a solution and adjacency of solutions can be tested in polynomial time, which holds for all aforementioned problems). One can easily construct artificial instance classes of reconfiguration problems such that the problem is in P, but has exponential diameter [3], but to our knowledge no natural examples are known. (That is, not constructed specifically to prove something about the reconfiguration problem at hand.)

With the goal of breaking one of these patterns, Kamiński et al [15, 16] introduced the SPR problem. Finding a shortest path can be done in polynomial time. Nevertheless, in [15, 16] examples were constructed where the solution graph has exponential diameter. This shows that regardless of whether SPR is in P or PSPACE-complete, one of the patterns is broken. The main open question from [16] was therefore that of determining the complexity of SPR.

In this paper, we answer that question by showing that SPR is PSPACE-complete. Therefore, this also answers the question posed in [12], by giving a rare example of a PSPACE-complete reconfiguration problem based on a decision problem in P. We remark that it is not the first example: in [3] it is shown that 4-Color Path is also PSPACE-hard for bipartite graphs. Since every bipartite graph is 2-colorable, the corresponding decision problem is trivial. Our PSPACE-completeness result is presented in Section 3. We remark that our PSPACE-completeness result, after it appeared in a preprint [2], has already proved its usefulness for showing PSPACE-completeness of other problems: in [17], the result has been applied to show that Independent Set Reconfiguration remains PSPACE-hard even when restricted to perfect graphs.

We give the following positive results on SPR. We show that when G is chordal or claw-free, SPR can be decided in polynomial time. A graph is *chordal* if it contains no induced cycles of length more than 3. This is a well-studied class of perfect graphs, which includes for instance k-trees and interval graphs [8]. A graph is *claw-free* if it contains no induced $K_{1,3}$ subgraph. This is again a well-studied graph class, see e.g. [7]. We also show that for these graph classes, the diameter of components of $\mathrm{SP}(G, s, t)$ is always linearly bounded. For chordal graphs, we can actually construct a *shortest* rerouting sequence in polynomial time. In contrast, in [15], it was shown that for general graphs, finding a shortest rerouting sequence is NP-hard, even for graph classes where there always exists one of polynomial length.

In the context of reconfiguration problems, other types of questions are commonly studied as well. Above, we considered the *reachability question*: can one given solution be reached from another given solution? The related *connectivity question* has also been well-studied [10, 4, 5, 9]: is the solution graph connected? For chordal graphs G, we show that $\mathrm{SP}(G, s, t)$ is always connected. If G is claw-free, we show that it can be decided in polynomial time whether $\mathrm{SP}(G, s, t)$ is connected. Our results on chordal graphs are presented in Section 4, and the results on claw-free graphs in Section 5. Because of space constraints, some (details of) proofs have been omitted.

2 Preliminaries

For graph theoretical notions not defined here, we refer to [8]. We will consider undirected and simple graphs throughout. A *walk* of length k from v_0 to v_k in a graph G is a vertex sequence $v_0, \dots, v_k$, such that for all $i \in \{0, \dots, k-1\}$, $v_i v_{i+1} \in E(G)$. It is a *path* if all vertices are distinct. It is a *cycle* if $k \geq 3$, $v_0 = v_k$, and $v_0, \dots, v_{k-1}$ is a path. With a path or cycle $W = v_0, \dots, v_k$ we associate a subgraph of G as well, with vertex set $V(W) = \{v_0, \dots, v_k\}$ and edge set $E(W) = \{v_i v_{i+1} \mid i \in \{0, \dots, k-1\}\}$. A path from s to t is also called an *st-path*. The *distance* from s to t is the length of a *shortest st-path*. The *diameter* of a graph is the maximum distance from s to t over all vertex pairs s, t.

A hypergraph $H = (V, E)$ consists of a vertex set V, and a set E of hyperedges, which are subsets of V. A walk in H of length k is a sequence of vertices $v_0, \dots, v_k$ such that for every i, there exists a hyperedge $e \in E$ with $\{v_i, v_{i+1}\} \subseteq e$. Using this notion of walks, connectivity and components of hypergraphs are defined the same as for graphs.

Throughout this paper, we will consider a graph G with vertices $s, t \in V(G)$. We will only be interested in shortest st-paths in G, and use d to denote their length. For $i \in \{0, \dots, d\}$, we define $L_i \subseteq V(G)$ to be the set of vertices that lie on a shortest st-path, at distance i from s. So $L_0 = \{s\}$, and $L_d = \{t\}$ (even if there may be more vertices at distance d of s). A set L_i is also called a *layer*. With respect to a given layer L_i, the *previous layer* is L_{i-1}, and the *next layer* is L_{i+1}. Clearly, if there is an edge $xy \in E(G)$ with $x \in L_i$ and $y \in L_j$, then $|j - i| \leq 1$. Note that a shortest st-path P contains exactly one vertex from every layer. For $i \in \{0, \dots, d\}$, this vertex will be called the *L_i-vertex* of P.

The graph G will be undirected, so we use the notation $N(v)$ to denote the set of neighbors of a vertex $v \in V(G)$. However, if $v \in L_i$, then we will use $N^-(v)$ to denote $N(v) \cap L_{i-1}$, and call these neighbors the *in-neighbors of* v. Similarly, $N^+(v)$ denotes $N(v) \cap L_{i+1}$, and these are called the *out-neighbors of* v.

Recall that a rerouting sequence from P to Q is a sequence $Q_0, \dots, Q_k$ of shortest st-paths with $Q_0 = P$, $Q_k = Q$, such that for every $j \in \{0, \dots, k-1\}$, Q_j and Q_{j+1} differ in at exactly one vertex. Let L_i be the layer in which they differ, and $u = L_i \cap V(Q_j)$ and $v = L_i \cap V(Q_{j+1})$. Then we also say that Q_{j+1} is obtained from Q_j with a rerouting step $u \to v$ in layer L_i.

3 PSPACE-Completeness

In this section we prove that the SPR problem is PSPACE-complete. A *k-color assignment* α for a graph G is a function $\alpha : V(G) \to \{1, \dots, k\}$. A *$k$-coloring* α for a graph G is a color assignment such that for all $uv \in E(G)$, $\alpha(u) \neq \alpha(v)$. For a given graph G, the *k-color graph* $\mathcal{C}_k(G)$ has vertex set consisting of all k-colorings of G, where two colorings are adjacent if they differ only in one vertex. A walk in $\mathcal{C}_k(G)$ from α to β will also be called a *recoloring sequence* from α to β. The problem *k-Color Path* has as input a graph G, with two k-colorings α and β. The question is whether there exists a walk from α to β in $\mathcal{C}_k(G)$. k-Color Path has been shown to be PSPACE-complete in [3].

Let G, α, β be an instance of 4-Color Path, with $V(G) = \{v_1, \dots, v_n\}$. We will now describe how to construct an equivalent SPR instance G' with two shortest st-paths P_α and P_β. Every shortest st-path in G' will correspond to a 4-color assignment for G (though not necessarily a 4-coloring!). To indicate this correspondence, some vertices of G' will be colored with the four colors $\{1, 2, 3, 4\}$. The other vertices will be colored with a fifth color, namely *black*. Note that this 5-color assignment for G' will not be a coloring of G'. G' will consist of one *main strand*, which contains the paths P_α and P_β, and $6n$ *recoloring strands*: one for every combination of a vertex $v_i \in V(G)$ and two colors $\{c_1, c_2\} \subset \{1, 2, 3, 4\}$.

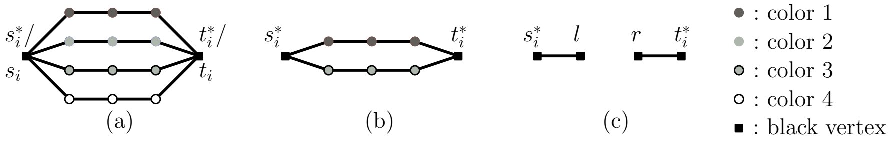


Fig. 1. Different variants of the gadgets H_i and H_i^* used in the construction

The construction of G' starts by introducing the vertices s and t. The *main strand* is constructed as follows. For each $v_i \in V(G)$, introduce a vertex gadget H_i as shown in Figure 1 (a). The leftmost vertex of H_i is labeled s_i, and the rightmost vertex t_i. These vertices are colored black. H_i consists of four disjoint s_it_i-paths of length 4, one for each color. All internal vertices of the paths are colored in the color assigned to the path. The four vertices of H_i that are neither adjacent to s_i nor to t_i are called *middle vertices* of H_i. These gadgets H_i are connected as follows: add edges ss_1 and t_nt, and for every $i \in \{1,\ldots,n-1\}$, add an edge t_is_{i+1}. At this point the graph is connected, and every vertex lies on a shortest st-path. Observe that the distance from s to s_i (t_i) is $5i-4$ (resp. $5i$), and the distance from s to t is $5n+1$. Hence this defines for every vertex which layer L_i it is part of, with $i \in \{0,\ldots,5n+1\}$.

We now define the *recoloring strands*. For each $v_i \in V(G)$ and each color pair $\{c_1,c_2\} \subset \{1,2,3,4\}$, we introduce a recoloring strand called the $v_i,\{c_1,c_2\}$-*strand*, defined as follows. Let $\{c_3,c_4\} = \{1,2,3,4\}\backslash\{c_1,c_2\}$. First we introduce gadgets H^*_j for every $j \in \{1,\ldots,n\}$. (Whenever we mention gadgets H^*_j or vertices s^*_j, t^*_j, l and r below, this refers to the gadgets and vertices for the given $v_i,\{c_1,c_2\}$-strand.)

- If $j \neq i$ and $v_iv_j \notin E(G)$, then define H^*_j to be isomorphic to H_j (see Figure 1 (a)), with the same 5-color assignment. The leftmost and rightmost (black) vertices are now labeled s^*_j and t^*_j respectively.
- If $j \neq i$ and $v_iv_j \in E(G)$, then define H^*_j to be as shown in Figure 1 (b). The leftmost and rightmost (black) vertices are labeled s^*_j and t^*_j again. Now there are only two disjoint paths from s^*_j to t^*_j, which are colored with the colors c_3 and c_4.
- H^*_i is the gadget shown in Figure 1 (c). Here s^*_i has one neighbor labeled l, and t^*_i has one neighbor labeled r.

Complete the strand by adding edges ss^*_1, t^*_nt and $t^*_js^*_{j+1}$ for every $j \in \{1,\ldots, n-1\}$. Note that if we add edges from l and r to a main strand vertex in layer L_{5i-2}, which we will do below, then all vertices of the new strand lie on st-paths of length $5n+1$ as well, and no shorter st-paths have been created. This defines for every vertex in the new strand which distance layer it is part of. We will refer to these layers in the next step, where we show how to connect the vertices of this recoloring strand to the main strand; see Figure 2. For all $j < i$, add the following edges: Add edges between s^*_j and every main-strand vertex in the next layer that has a color that is also used in H^*_j. Next, for every non-black vertex v of H^*_j, add an edge between v and the main-strand vertex in the next layer that has the same color as v, or is black. Finally, add an edge $t^*_js_{j+1}$. For all $j > i$, edges between H^*_j and the main strand are added similarly, except that vertices of H^*_j are connected to vertices in the *previous* layer. (See H^*_4 in Figure 2 for an example.) For H^*_i we add edges as follows: Connect s^*_i (t^*_i) to the main strand vertices in the next (resp. previous) layer with colors c_1 and c_2. Finally, connect both remaining vertices l and r of H^*_i to both middle vertices of H_i that have colors c_1 and c_2. Introducing such a $v_i,\{c_1,c_2\}$-*strand* for every $v_i \in V(G)$ and $\{c_1,c_2\} \subset \{1,2,3,4\}$ completes the construction of G'.

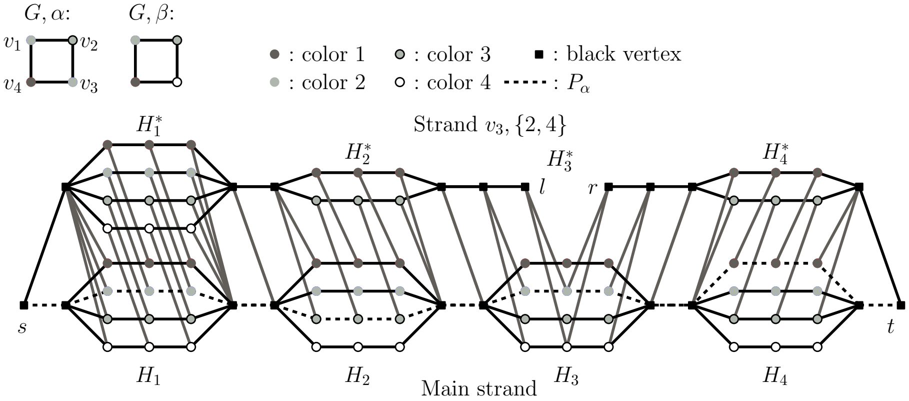


Fig. 2. A k-Color Path instance G, α, β, and two strands of the resulting graph G'

We now show how to construct a path P_γ for any given 4-coloring γ of G; see Figure 2. The path P_γ contains only main strand vertices. Since it should be a shortest st-path, it contains exactly one vertex of every layer. Every layer contains vertices of a unique gadget H_i of the main strand. In the case that the layer contains a single black vertex from H_i, this is the vertex that is included in P_γ. In the case that the layer contains vertices of colors $1, \ldots, 4$ of H_i, use the vertex of color $\gamma(v_i)$ for P_γ. This way, we define the paths P_α and P_β, using the given colorings α and β, respectively.

The purpose of the recoloring strands is as follows: Consider two adjacent colorings α and β. Suppose they differ only in vertex v_i, where their respective colors are $c_1 = \alpha(v_i)$ and $c_2 = \beta(v_i)$. Then all neighbors of v_i are colored with colors in $\{1, 2, 3, 4\} \backslash \{c_1, c_2\}$. Therefore, P_α can be rerouted to a shortest st-path P' that lies entirely in the $v_i, \{c_1, c_2\}$-strand, except for the vertex in layer L_{5i-2}. This rerouting is done by making rerouting steps in layers $L_1, \ldots, L_{5i-1}$ in increasing order, and subsequently in layers $L_{5n}, \ldots, L_{5i-3}$ in decreasing order. Note that the path P' must use vertices that have the same color as the P_α-vertex of the same layer. Then, with a single rerouting step, the color of the vertex in layer L_{5i-2} can be changed from c_1 to c_2. Rerouting the path back to the main strand, in reversed layer order, then gives P_β. This can be done for every pair of adjacent colorings, so a recoloring sequence from α to β gives a rerouting sequence from P_α to P_β. This yields:

Lemma 1. *If there is a recoloring sequence for G from α to β, then there is a rerouting sequence from P_α to P_β for G'.*

With any shortest st-path P' in G', we associate a color assignment where $v_i \in V(G)$ receives the same color as the (middle) vertex in $L_{5i-2} \cap V(P')$. The converse of Lemma 1 can then be proved as follows: a rerouting step can only change the L_{5i-2}-vertex of a shortest st-path P' from a vertex of color c_1 to a vertex of color c_2 if the vertices of P' in the previous and next layer are part of the $v_i, \{c_1, c_2\}$-strand. In that case, all vertices of P' except the L_{5i-2}-vertex must be part of this strand, which shows that P' corresponds to a 4-color assignment in which the neighbors of v_i are not colored with c_1 or c_2. Hence if P' corresponds to a 4-coloring, then the shortest st-path resulting from the rerouting step corresponds again to a 4-coloring. Therefore, any rerouting sequence from P_α to P_β can be mapped to a recoloring sequence from α to β.

Lemma 2. *If there is a rerouting sequence from P_α to P_β for G', then there is a recoloring sequence in G from α to β.*

Theorem 3. *SPR is PSPACE-complete.*

Proof: 4-Color Path is PSPACE-complete [3]. Our transformation from G to G' is polynomial. By Lemma 1 and 2, G, α, β is a YES-instance for 4-Color Path if and only if G', P_α, P_β is a YES-instance for SPR. This proves PSPACE-hardness. Membership in PSPACE follows from Savitch's Theorem [19] which states that PSPACE = NPSPACE; the problem is easily seen to be in NPSPACE. □

4 Chordal Graphs

We will show in this section that for chordal graphs G, the SPR problem can be decided in polynomial time. In fact, we prove that if G is chordal, then $\mathrm{SP}(G, s, t)$ is connected and has diameter at most $d-1$, where d is the distance from s to t.

Theorem 4. *Let G be a chordal graph, and let P and Q be two shortest st-paths in G, of length d. Then a rerouting sequence from P to Q exists, of length at most $|V(P)\backslash V(Q)| \leq d-1$.*

Proof sketch: We prove the statement by induction over $c = |V(P)\backslash V(Q)|$. The case $c = 0$ is obvious, so now assume that $c \geq 1$. Let $P = u_0, u_1, \ldots, u_d$, and $Q = v_0, v_1, \ldots, v_d$. Let i be the lowest index such that $u_i \neq v_i$. Let j be the lowest index with $j > i$ such that $u_j = v_j$. If $j = i+1$, then both u_i and v_i are adjacent to both u_{i-1} and u_{i+1}, so to P we can apply the rerouting step $u_i \to v_i$, to obtain a new shortest st-path P' in G that has one more vertex in common with Q. So by induction, the distance from P' to Q in $\mathrm{SP}(G, s, t)$ is at most $c-1$. Then the distance from P to Q is at most c. This proves the statement in the case $j = i+1$, so now assume that $j \geq i+2$.

Note that then $C = u_{i-1}, u_i, \ldots, u_{j-1}, u_j, v_{j-1}, \ldots, v_i, v_{i-1}$ is a cycle in G, of length at least 6. Using C and using that G is chordal, it can be shown that G contains an edge e with $e = u_i v_{i+1}$ or $e = u_{i+1} v_i$. If $e = u_{i+1} v_i$, then both u_i and v_i are adjacent to both u_{i-1} and u_{i+1}, so to P we may apply the rerouting step $u_i \to v_i$, to obtain a new shortest st-path P' that has at least one more vertex in common with Q. Then the proof can be concluded the same as before. The remaining case where $e = u_i v_{i+1}$ is symmetric; a rerouting step $v_i \to u_i$ can be applied to Q. □

The above proof gives a polynomial time algorithm for constructing the rerouting sequence. Obviously a rerouting sequence from P to Q requires at least $|V(P)\backslash V(Q)|$ rerouting steps, so we may conclude:

Corollary 5. *Let G be a chordal graph with shortest st-paths P and Q. In polynomial time, a shortest rerouting sequence from P to Q can be constructed.*

5 Claw-Free Graphs

In this section we show that deciding SPR, and deciding whether $\mathrm{SP}(G, s, t)$ is connected can both be done in polynomial time in the case where G is claw-free. A *claw* is a $K_{1,3}$ graph. A graph G is *claw-free* if it contains no claws as induced subgraphs. In other words, G is not claw-free if and only if it contains a subgraph H that consists of one vertex c of degree 3, and three leaves l_1, l_2, l_3, such that the leaves are pairwise nonadjacent in G. Such an *induced* subgraph will be called a *c-claw with leaves* l_1, l_2, l_3 for short.

Let $u \in L_i$. We say that u *has maximal in-neighborhood* if there is no $v \in L_i$ with $N^-(u) \subset N^-(v)$. (Note that we distinguish between subset $\subseteq$ and strict subset $\subset$.) In that case, $N^-(u)$ is a *maximal in-neighborhood in* L_{i-1}. These notions are defined analogously for out-neighborhoods. With a layer L_i, we associate the following hypergraph $\mathcal{H}_i$: $\mathcal{H}_i$ has vertex set L_i, and the edges correspond to the maximal in-neighborhoods in L_i. So for every $e \in E(\mathcal{H}_i)$, there exists a vertex $a \in L_{i+1}$ with $N^-(a) = e$.

The main result of this section is proved as follows. We first give some simple reduction rules. These are based on the fact that it is safe to delete a vertex v, if we know that it is not part of any shortest st-path that can be reached from the given shortest st-path P. We give two ways to identify such vertices. For reduced, claw-free SPR instances G', P, Q that do not have such vertices, we actually show that $\mathrm{SP}(G', s, t)$ is connected. Proposition 6 follows from this observation: whenever a rerouting step $x \to y$ in layer L_i is made, there is a vertex $z \in L_{i+1}$ with $x, y \in N^-(z)$, so x and y are in the same component of $\mathcal{H}_i$.

Proposition 6. *Let P be a shortest st-path in a graph G. For every shortest st-path Q that is reachable from P in $SP(G, s, t)$ and every i, the L_i-vertex is part of the same component of $\mathcal{H}_i$ as the L_i-vertex of P.*

Proposition 7. *Let P be a shortest st-path of length d in a claw-free graph G, in which every vertex lies on a shortest st-path. For every shortest st-path Q that is reachable from P in $SP(G, s, t)$ and every $i \in \{2, \dots, d-2\}$, the L_i-vertex of Q is adjacent to the L_i-vertex of P.*

Proof: Consider a rerouting sequence $Q_0, \dots, Q_k$ from $Q_0 = P$ to $Q_k = Q$, and let x_j be the L_i-vertex of Q_j, for every $j \in \{0, \dots, k\}$. Assume that the claim is not true, so then we may choose ℓ to be the lowest index such that $x_0 x_\ell \notin E(G)$.

If x_0 and x_ℓ have a common neighbor z in either L_{i-1} or L_{i+1}, then a z-claw with leaves x_0, x_ℓ and y exists, for some vertex $y \in L_{i-2}$ or $y \in L_{i+2}$, respectively.

(z has such a neighbor y since it lies on a shortest st-path.) So since G is claw-free, we may conclude that $N^-(x_0) \cap N^-(x_\ell) = \emptyset$, and $N^+(x_0) \cap N^+(x_\ell) = \emptyset$. If $x_{\ell-1}$ has a neighbor $y \in L_{i-1} \backslash N^-(x_0)$ and a neighbor $z \in L_{i+1} \backslash N^+(x_0)$, then an $x_{\ell-1}$-claw with leaves x_0, y, z exists. So w.l.o.g. we may assume that $N^-(x_{\ell-1}) \subseteq N^-(x_0)$. But then $N^-(x_{\ell-1}) \cap N^-(x_\ell) = \emptyset$, which contradicts that a rerouting step $x_{\ell-1} \to x_\ell$ is possible. □

Definition 8. *Let G be a claw-free graph with vertices s and t at distance d from each other. Then G is called st-*reduced *if*

- *all vertices lie on a shortest st-path,*
- *for every $i \in \{1, \ldots, d-1\}$, $\mathcal{H}_i$ is connected, and*
- *for every $i \in \{2, \ldots, d-2\}$, L_i is a clique.*

Propositions 6 and 7 can be used to identify in polynomial time vertices that are not part of any shortest st-path that is reachable from a given path P. Deleting these, together with vertices not on shortest st-paths, gives an st-reduced instance.

Lemma 9. *Let G be a claw-free graph, with shortest st-path P. In polynomial time, we can construct an induced claw-free subgraph G' such that (i) G' is st-reduced, and (ii) a shortest st-path Q of G is reachable from P in $SP(G, s, t)$ if and only if $V(Q) \subseteq V(G')$, and Q is reachable from P in $SP(G', s, t)$.*

The last property from Definition 8 shows that every pair of vertices in one layer is adjacent; this makes it much easier in our proofs to obtain a contradiction by exhibiting an induced claw. We use this to prove Lemmas 10 and 11.

Lemma 10. *Let P be a shortest st-path of length d in a claw-free st-reduced graph G. In polynomial time, a rerouting sequence of length at most $d-1$ can be constructed, from P to a shortest st-path P' in which every vertex has maximal out-neighborhood. The same holds for the case of maximal in-neighborhoods.*

Proof: Let $P = u_0, u_1, \ldots, u_{d-1}, u_d$. Define $v_0 := u_0 (= s)$. For $i = 1, \ldots, d-1$, in increasing order, we change the L_i-vertex u_i of P as follows. If the out-neighborhood of u_i is not maximal, then choose $v_i \in L_i$ with $N^+(u_i) \subset N^+(v_i)$, and $N^+(v_i)$ maximal. If possible, choose v_i such that $v_i \in N^+(v_{i-1})$. Then, apply the rerouting step $u_i \to v_i$. If u_i already has maximal out-neighborhood then simply define $v_i = u_i$.

It remains to show that $u_i \to v_i$ is in fact a rerouting step. By definition, $u_{i+1} \in N^+(v_i)$, so the L_{i+1}-vertex of the current path $v_0, \ldots, v_{i-1}, u_i, u_{i+1}, \ldots, u_d$ poses no problem. It might however be that v_i is not adjacent to v_{i-1}. In that case, $i \geq 2$. Choose a vertex $x \in N^-(v_i)$. Since $v_i \in N^+(x) \backslash N^+(v_{i-1})$, but $N^+(v_{i-1})$ is maximal, there exists at least one $y \in N^+(v_{i-1}) \backslash N^+(x)$. By choice of v_i, there exists at least one $z \in N^+(v_i) \backslash N^+(y)$, otherwise y has maximal out-neighborhood as well, and we would have chosen $v_i = y$ (since we gave preference to out-neighbors of v_{i-1}). This however gives a v_i-claw with leaves x, y, z, a contradiction. The in-neighborhood case is analog. □

Maximal in- and out-neighborhoods are required to apply the next lemma, which is concerned with rerouting a single layer L_i.

Lemma 11. *Let G be a claw-free, st-reduced graph, with distance d between s and t. Let $P = u_0, \ldots, u_d$ be a shortest st-path. Let $i \in \{1, \ldots, d-1\}$ such that u_{i-1} has maximal out-neighborhood and u_{i+1} has maximal in-neighborhood. Then for every $w \in N^+(u_{i-1})$, using at most $2|L_i|$ rerouting steps, P can be modified to a shortest st-path $P' = v_0, \ldots, v_d$ with $v_i = w$, and $v_j = u_j$ for all $j \in \{0, \ldots, d\} \backslash \{i, i+1\}$.*

Proof sketch: We assume $1 \leq i \leq d-2$, since the case $i = d-1$ is trivial. Consider a shortest path $x_0, \ldots, x_k$ in $\mathcal{H}_i$ from u_i to w, so $x_0 = u_i$ and $x_k = w$. (This exists since G is st-reduced.) For $j \in \{1, \ldots, k\}$, let $a_j \in L_{i+1}$ be a vertex with maximal in-neighborhood such that $\{x_{j-1}, x_j\} \in N^-(a_j)$. (Such a vertex exists by the definition of the hypergraph $\mathcal{H}_i$.) Choose $a_1 = u_{i+1}$ if u_{i+1} satisfies this condition. In addition, let $a_0 = u_{i+1}$. The rerouting sequence from P to P' uses the following rerouting steps: If $a_0 \neq a_1$, then first replace $a_0 = u_{i+1}$ by a_1. Next, replace $x_0 = u_i$ by x_1. Next, replace a_1 by a_2, and x_1 by x_2. Continue making rerouting steps alternatingly in layer L_{i+1} and L_i until x_{k-1} can be replaced by $x_k = w$, to obtain the desired path P'. Note that by definition of the x_j and a_j vertices, at every point the L_i- and L_{i+1}-vertex are adjacent. It may however be that at some point, a vertex x_j is not adjacent to u_{i-1}. In this case, the aforementioned rerouting sequence is preceded by replacing u_{i-1} by a vertex $y \in N^-(x_j)$ with maximal out-neighborhood, and succeeded by replacing y by u_{i-1} again. Using the fact that G is claw-free and st-reduced, it can be shown that this way, a shortest st-path is maintained throughout. □

Combining Lemmas 10 and 11 gives the main combinatorial result, for st-reduced claw-free graphs. The main algorithmic results, Theorem 13 and 14, follow easily. Their proofs are similar. We leave the second proof as an exercise.

Theorem 12. *Let G be an st-reduced claw-free graph on n vertices, with distance d from s to t. Between any two shortest st-paths P and Q in G, a rerouting sequence of length at most $2n + 2d - 6$ exists.*

Proof: First apply at most $d-1$ rerouting steps to P to obtain a shortest st-path P' in which every vertex has a maximal in-neighborhood (Lemma 10). Similarly, apply at most $d-1$ rerouting steps to Q to obtain a shortest st-path Q' in which every vertex has a maximal out-neighborhood (Lemma 10).

Now P' can be modified to Q' in $d-1$ stages i, with $i \in \{1, \ldots, d-1\}$. Denote $P_0 = P' = u_0, \ldots, u_d$, and $Q' = v_0, \ldots, v_d$. At the start of the ith stage, we have a shortest st-path $P_{i-1} = v_0, \ldots, v_{i-1}, a, u_{i+1}, \ldots, u_d$ for some $a \in L_i$ (note that for $i = 1$, P_0 is of this form). Using at most $2|L_i|$ rerouting steps, P_{i-1} can be modified into a shortest st-path $P_i = v_0, \ldots, v_i, a', u_{i+2}, \ldots, u_d$ for some $a' \in L_{i+1}$. This follows from Lemma 11.

After $d-1$ stages, this procedure terminates with a path $v_0, \ldots, v_{d-1}, u_d$, which equals Q'. The total number of rerouting steps for these stages is at most $\sum_{i \in \{1, \ldots, d-1\}} 2|L_i| = 2(n-2)$. In total, this shows that P and Q can both be rerouted to a common shortest st-path Q', in at most $2(n-2)+(d-1)$ and $d-1$ steps, respectively. Combining these rerouting sequences gives a rerouting sequence from P to Q of length at most $2n+2d-6$. □

Theorem 13. *Let G be a claw-free graph on n vertices, and let P and Q be two shortest st-paths in G, of length d. In polynomial time it can be decided whether Q is reachable from P in SP(G,s,t), and if so, a rerouting sequence of length at most $2n+2d-6$ exists.*

Proof: By Lemma 9, in polynomial time we can construct an st-reduced induced subgraph G' of G such that any shortest st-path Q' is reachable from P in G if and only if it is reachable from P in G'. So if Q is not a shortest st-path of G' (at least one of its vertices was deleted), we may conclude it is not reachable. Otherwise, Theorem 12 shows that Q is reachable from P, with a rerouting sequence of length at most $2|V(G')|+2d-6 \leq 2n+2d-6$. □

Theorem 14. *Let G be a claw-free graph on n vertices. In polynomial time it can be decided whether SP(G,s,t) is connected.*

6 Discussion

We showed that SPR is PSPACE-complete, which is somewhat surprising since the problem of finding shortest paths is easy. Nevertheless, our results otherwise confirm the typical behavior of reconfiguration problems: for instances where we can decide SPR in polynomial time (chordal and claw-free graphs), the diameter is polynomially bounded – in this case, even linearly bounded. In addition, for these graph classes it can be decided efficiently whether SP(G,s,t) is connected. The main question that is left open here is: *What is the complexity of deciding whether SP(G,s,t) is connected, for general graphs G?*

We note that for the SPR instances G', P_α, P_β constructed in Section 3, the proof of Lemma 2 shows that SP(G',s,t) is always disconnected (unless the 4-Color Path instance G has no edges).

We showed that for chordal graphs G, one can even find *shortest* rerouting sequences in polynomial time. Is this possible for claw-free graphs as well? To be precise, for two shortest st-paths P and Q in a claw-free graph G and $k \in \mathbb{N}$, can it be decided in polynomial time whether a rerouting sequence from P to Q of length at most k exists, or is this problem NP-complete? Recall that for general graphs, the NP-hardness of finding a shortest rerouting sequence was proved in [15]. By our linear diameter result, this (decision) problem lies in NP for claw-free graphs. Finally, it is interesting to search for other graph classes for which SPR can be solved in polynomial time. Graphs of bounded treewidth form a prime candidate.

Acknowledgements. The author would like to thank Paul Medvedev and Martin Milanič for the introduction to this problem, and for comments on an earlier version of this paper. This research was supported by DFG grant BO 3391/1-1.

References

1. Bonamy, M., Johnson, M., Lignos, I., Patel, V., Paulusma, D.: On the diameter of reconfiguration graphs for vertex colourings. Electronic Notes in Discrete Mathematics 38, 161–166 (2011)
2. Bonsma, P.: Shortest path reconfiguration is PSPACE-hard. arXiv:1009.3217v1 (2010)
3. Bonsma, P., Cereceda, L.: Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. Theoret. Comput. Sci. 410(50), 5215–5226 (2009)
4. Cereceda, L., van den Heuvel, J., Johnson, M.: Connectedness of the graph of vertex-colourings. Discrete Appl. Math. 308(5-6), 913–919 (2008)
5. Cereceda, L., van den Heuvel, J., Johnson, M.: Mixing 3-colourings in bipartite graphs. European J. Combin. 30(7), 1593–1606 (2009)
6. Cereceda, L., van den Heuvel, J., Johnson, M.: Finding paths between 3-colorings. J. Graph Theory 67(1), 69–82 (2011)
7. Chudnovsky, M., Seymour, P.: The structure of claw-free graphs. In: Surveys in Combinatorics 2005, pp. 153–171. Cambridge Univ. Press (2005)
8. Diestel, R.: Graph theory, 4th edn. Graduate Texts in Mathematics, vol. 173. Springer, Berlin (2010)
9. Eggermont, C.E.J., Woeginger, G.J.: Motion planning with pulley, rope, and baskets. In: STACS 2012. LIPIcs, vol. 14, pp. 374–383 (2012)
10. Gopalan, P., Kolaitis, P.G., Maneva, E., Papadimitriou, C.H.: The connectivity of boolean satisfiability: Computational and structural dichotomies. SIAM J. Comput. 38(6) (2009)
11. Hearn, R.A., Demaine, E.D.: PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. Theoret. Comput. Sci. 343(1-2), 72–96 (2005)
12. Ito, T., Demaine, E.D., Harvey, N.J.A., Papadimitriou, C., Sideri, M., Uehara, R., Uno, Y.: On the Complexity of Reconfiguration Problems. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 28–39. Springer, Heidelberg (2008)
13. Ito, T., Demaine, E.D., Harvey, N.J.A., Papadimitriou, C.H., Sideri, M., Uehara, R., Uno, Y.: On the complexity of reconfiguration problems. Theoret. Comput. Sci. 412(12-14), 1054–1065 (2011)
14. Ito, T., Kamiński, M., Demaine, E.D.: Reconfiguration of List Edge-Colorings in a Graph. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 375–386. Springer, Heidelberg (2009)
15. Kamiński, M., Medvedev, P., Milanič, M.: Shortest paths between shortest paths. Theoret. Comput. Sci. 412(39), 5205–5210 (2011)
16. Kamiński, M., Medvedev, P., Milanič, M.: Shortest Paths between Shortest Paths and Independent Sets. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2010. LNCS, vol. 6460, pp. 56–67. Springer, Heidelberg (2011)
17. Kamiński, M., Medvedev, P., Milanič, M.: Complexity of independent set reconfigurability problems. Theoret. Comput. Sci. 439, 9–15 (2012)
18. Papadimitriou, C.H., Schäffer, A.A., Yannakakis, M.: On the complexity of local search. In: STOC 1990, pp. 438–445. ACM, New York (1990)
19. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. System. Sci. 4, 177–192 (1970)

Computing with Large Populations Using Interactions

Olivier Bournez[1,⋆], Pierre Fraigniaud[2,⋆⋆], and Xavier Koegler[2]

[1] LIX, Ecole Polytechnique and CNRS, France
[2] LIAFA, CNRS and University Paris Diderot, France

Abstract. We define a general model capturing the behavior of a population of anonymous agents that interact in pairs. This model captures some of the main features of opportunistic networks, in which nodes (such as the ones of a mobile ad hoc networks) meet sporadically. For its reminiscence to Population Protocol, we call our model *Large-Population Protocol*, or LPP. We are interested in the design of LPPs enforcing, for every $\nu \in [0, 1]$, a proportion ν of the agents to be in a specific subset of marked states, when the size of the population grows to infinity; In which case, we say that the protocol *computes* ν. We prove that, for every $\nu \in [0, 1]$, ν is computable by a LPP if and only if ν is algebraic. Our positive result is constructive. That is, we show how to construct, for every algebraic number $\nu \in [0, 1]$, a protocol which computes ν.

1 Introduction

1.1 Motivation

So-called *opportunistic networks* (see, e.g., [32]) are characterized by connections between users that appear sporadically, which are as many opportunities for exchanging data or forwarding messages. As such, they form a subclass of the so-called delay-tolerant networks (DTNs). A typical and probably prominent example of opportunistic networks is sparse mobile ad hoc networks, as analyzed in, e.g., the Zebra project [29], as well as in several other projects aiming at understanding the potential of opportunistic networks [15,27]. This paper is interested in the computing power of such networks.

More specifically, we are focussing on the *slicing* problem [28], which refers to the ability of creating a virtualized network running over multiple physical nodes, where the nodes are partitioned in multiple *slices*. Many metrics may be used to sort the nodes for assigning them to different slices. Typical metrics are available resources, such as memory, bandwidth, or computing power. However, as underlined in [28], slicing the network by focusing only on the size of the slices, also deserves to be investigated, for applications to systems involving devices

[⋆] Supported by ANR project SHAMAN.

[⋆⋆] The last two authors are supported by ANR projects DISPLEXITY and PROSE, and by INRIA project GANG.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 234–246, 2012.

with similar resource capabilities. Independently from the context, we stress that the slice sizes are usually expressed as a percentage of the network size. Slicing algorithms in the literature are usually designed for peer-to-peer systems in which powerful peers are capable of generating random numbers [28], or taking advantage of a storage capacity proportional to the population size [23] or to the size of the neighborhood [22]. In this paper, we show how to slice the nodes of opportunistic networks composed of tiny and simple devices, *deterministically*, using only the basic resources of *finite state* agents, by taking benefits of the randomized interactions between these agents.

To illustrate our objective, consider a population of nodes (e.g., sensors) moving in a somewhat restricted environment. Assume moreover, for the sake of simplifying the presentation and the analysis, that nodes meet uniformly at random over time. Let us say one wants to slice these nodes into two slices of equal size. The following trivial 2-state protocol achieves this. Nodes are either in state positive or negative, and whenever two nodes meet, one of them becomes positive while the other becomes negative. Nodes then tend over time to partition themselves into two slices of equal size, when the size of the population grows to infinity. As a more sophisticated example, assume that one wants to slice the nodes in order to construct a slice such that the probability that two nodes in this slice meet is 1/2. That is, one wants to slice the nodes into two slices, with one slice amounting to a ratio $1/\sqrt{2}$ of the total number of nodes. This is ensured by the following 2-state protocol. Whenever two nodes meet, their actions depend on their current states + or −: if the nodes are in different states then both become positive; otherwise, one of them becomes positive while the other becomes negative. This dynamic can be summarized by the four transition rules:

$$\begin{array}{ccccccc} +- & \rightarrow & ++ & \qquad & ++ & \rightarrow & +- \\ -+ & \rightarrow & ++ & \qquad & -- & \rightarrow & +- \end{array} \tag{1}$$

This protocol has been extensively analyzed in [14], where it is proved that the proportion of positive nodes does converge to the desired ratio $1/\sqrt{2}$ over time, when the size of the population grows to infinity.

In some sense, the former protocol *computes* 1/2, while the latter protocol computes $1/\sqrt{2}$. One may of course be interested in computing other values. To start with, beside 1/2, is it possible to compute every rational in $[0,1]$? And beside rationals, is the protocol for $1/\sqrt{2}$ extendable to ratios of the form $x^{-1/k}$ for every $x \geq 1$ and $k \geq 2$? More generally, what is the limit of such protocols in term of their computing power? For instance, is it possible to compute solutions of trigonometric equations? E.g., can we design a protocol insuring that, asymptotically, a ratio $\frac{\pi}{4}$ of the nodes are in some prescribed state?

1.2 Framework

In order to determine the computing power of a collection of nodes such as the ones involved in an opportunistic network, we abstract our model from the specific technological constraints to be faced when one is dealing with networks (security, forwarding mechanism, mobility, etc.). In fact, a network applying a protocol such as the one in Eq. 1 can be considered as a *population protocol*, in the spirit of the model introduced in [3]. Essentially, in population protocols, nodes (or agents) are anonymous, their interactions are supposed to be scheduled so as to satisfy some natural fairness property, and their individual actions are independent from the size of the population.

Classical population protocols are however designed to compute *predicates* over their input configuration whereas the protocol in Eq. 1 computes a (real) value (i.e., the proportion of agents in positive state), independently from the initial configuration. Another difference with classical population protocols is that the result of our computation is asymptotic, when the size of the population grows to infinity. These differences can be tackled by considering a setting that we call *Large-Population Protocols* (LPPs), which is essentially population protocols whose behaviors are analyzed when the population grows to infinity. In this context, for any given number $\nu \in [0,1]$, a LPP is said to *compute* ν if the proportion of agents in some specific states, say the *marked* states, converges over time to ν when the population grows to infinity.

We can now reformulate the question raised in the previous section slightly more formally by: what can be computed by a LPP? More precisely, we address the problem of determining which numbers can be computed by Large-Population Protocols. That is, we are aiming at identifying the set of real numbers $\nu \in [0,1]$ for which there exists a LPP computing ν.

1.3 Our Results

We first define formally our model for Large-Population Protocols (LPP). The model is quite general in the sense that it encompasses all the models involving a "population" of agents, whenever they are dealing with anonymous agents that interact in pairs. We then prove that the execution of any LPP is well characterized by the behavior of a differential system. This characterization can be considered similar to what is usually done in mean field theory. However, we go beyond the simple application of mean field analysis by completely formalizing the connection between the execution of a LPP and the behavior of the corresponding differential system. Specifically, fix any protocol $\mathcal{P}$, and define $X = X(n,t)$ as the random variables equal to the proportion of agents in marked state at time t in a population of size n, during the execution of $\mathcal{P}$. We characterize the exact behavior of X when the size of population grows to infinity. Essentially, we prove that, whenever the initial state is close enough to a stable equilibrium, we have

$$X(n,t) \approx f(t) + \frac{1}{\sqrt{n}}\,\mathcal{N}(0,\chi) \tag{2}$$

where f is the solution of the differential system corresponding to $\mathcal{P}$, $\mathcal{N}(0,\chi)$ is a centered gaussian with covariance matrix χ depending on $\mathcal{P}$, and $\approx$ denotes a convergence in law after an appropriate rescaling.

Using the correspondence between LPPs and differential systems, we characterize the real numbers that are computable by LPPs as being all *algebraic* numbers in $[0, 1]$. The fact that transcendental numbers cannot be computed by LPPs is a consequence of arguments from model theory (mainly Tarski's effective procedure for quantifier elimination over real closed fields). Our main result is a proof that all algebraic numbers can be computed by LPPs. Our proof is constructive, that is, for every algebraic number $\nu \in [0, 1]$ described by some polynomial P, with rational coefficients, and satisfying $P(\nu) = 0$, we show how to construct a LPP computing ν in the sense of Eq. 2. That is, $X(n, t)$ satisfies the relation of Eq. 2, with $\lim_{t\to\infty} f(t) = \nu$.

The algorithmic construction proceeds in four stages. The first stage consists in constructing, for every *rational* ν, a LPP computing ν. The second stage of the construction consists in a form of derandomization for LPP. More precisely, we show that every protocol involving *probabilistic* transition rules where each probability is rational can be transformed into a protocol involving solely *deterministic* transition rules. Hence, the remaining two stages involve protocols using probabilistic transition rules. The third stage is heavily based on our characterization of LPP using differential systems. We construct a differential system corresponding to a LPP, and admitting ν as an equilibrium. This system is obtained by identifying a multivariate polynomial $\tilde{P}$ such that $\tilde{P}(x_1, \ldots, x_k) = P(x_1)$ whenever $x_i = x_{i-1}x_1$, where k is the degree of P. Interestingly, $\tilde{P}$ is *not* P in which x^i would be replaced by x_i. Instead, $\tilde{P}$ is specifically designed so that to yield a differential system which admits ν as an equilibrium, that can be in turn transformed into a protocol with rational probabilistic transitions. The fourth and last stage of the construction involves stability. We show how to modify the construction of the third stage to enforce that ν becomes a *stable* equilibrium. This is achieved by carefully modifying the polynomial $\tilde{P}$ so that the Routh-Hurwitz stability criterion can be applied, while preserving the ability to translate the differential system into a LPP.

1.4 Related Work

The model that we consider in this paper captures the behavior of any *large* population of *indistinguishable* agents interacting *in pairs* in a *Markovian* manner. This framework includes many models from nature, physics, and biology (see, e.g., [31]). Several papers have already demonstrated the benefit of using an algorithmic approach for understanding such models (see, e.g., the recent papers [12,18,20]). Conversely, models from nature, physics, and biology can be viewed as alternative paradigms of computation (see, e.g., [1,11]).

Classical models for capturing the dynamics of populations include *Lotka-Volterra* dynamics for *predator-prey* models, *replicator* dynamics, and, more generally, all kinds of models from evolutionary game theory. In particular, it is known that a subclass of protocols designed in the context of evolutionary game theory correspond to Lotka Volterra dynamics [17], which are in turn known to be equivalent to replicator dynamics [26]. The connections between the dynamics of games and population protocols has been studied in [13,17].

Population protocols have been introduced in [3]. The model was designed to decide logic predicates, and predicates computable by classical population protocols have been characterized [3,4] as being precisely the semi-linear predicates, that is, those predicates on counts of input agents definable in first-order Presburger arithmetic. Variants of the original model considered so far include restrictions on communications [2,5], random interactions [3,8,7], and mediated interactions [30]. Various kinds of fault tolerance have also been considered for population protocols [6,21]. We refer to [9] for a comprehensive introduction to population protocols, and to [16,19] for the description of formal methods for verifying such protocols.

A few papers addressed the asymptotic behavior of population protocols, when the population size grows to infinity. In [24], a framework for translating certain subclasses of differential equation systems into practical protocols for distributed systems, assuming a large population, is described. In [17], the authors study the dynamics and the stability of (probabilistic) population protocols via ordinary differential equations. [14] proves that there exists a close relationship between, on the one hand, classical finite population protocols, and, on the other hand, models obtained by ordinary differential equations. The protocol computing $1/\sqrt{2}$ described in Eq. 1 has been thoroughly studied in [14] where convergence is proved using weak-convergence methods for stochastic processes. In [10], the authors address the issue of convergence speed. It is proved that it is possible to compute $1/\sqrt{2}$ with arbitrary precision $\epsilon > 0$ in a time polynomial in $1/\epsilon$, using a number of agents polynomial in $1/\epsilon$.

2 Large-Population Protocols

In this section, we define Large-Population Protocols (LPP), and state formally what is meant by computing with LPPs. The general idea of the model has been introduced in [10] and [14]. The following subsection recalls the main features of the model. Our first contribution is a formal specification of the asymptotic behavior of a LPP.

2.1 The Model

We consider a population of n anonymous agents, each of which can be in finitely many possible states, from a finite set Q. This population evolves with time. We assume a synchronous discrete-time system, and, at each round, two agents a and b are selected among the n agents. The selection is performed uniformly at random, independently from the past. Note that the original population protocol model [3] just assumes a specific fairness hypothesis for the interactions between the agents, which are under the control of an adversary with restricted power. Nevertheless, when the size of the population goes to infinity, uniform sampling of agents appears to be a natural way to extend the fairness hypothesis used in classical population protocols. Moreover, uniform sampling is consistent with the interpretation of agents as autonomous entities moving at random. (See also [9] for a discussion on the random adversary in finite state systems.)

The two agents a, b that are selected, can interact and change their states according to a set Δ of transition rules of the form $q_i\ q_j \to q_k\ q_l$ where $(q_k, q_l) = \Delta(q_i, q_j)$.

Note that the transition is not necessarily symmetric, i.e., the selected pair (a, b) may cause a transition different from the one caused by the pair (b, a). In other words, we do not necessarily assume $\Delta(q_i, q_j) = \Delta(q_j, q_i)$. Let us identify a specific subset Q^+ of states of Q, say $\{q_1, q_2, \cdots, q_m\}$, to be the *marked state*, and denote $Q = \{q_1, q_2, \cdots, q_{|Q|}\}$. The pair (Q, Δ) entirely defines a protocol $\mathcal{P}$. Such a protocol is called *large-population protocol* because, informally, we will say that $\mathcal{P}$ computes some given number ν if $\mathcal{P}$ enforces the proportion of agents in states $q \in Q^+$ to converge to ν along with time, when the size n of the population grows to infinity.

To get some intuition of how to formally define computation, assume first that n is fixed, and assume $m = 1$. The evolution (with time) of the population can be modeled by a discrete-time homogeneous Markov chain whose states are all the possible configurations of the system. For the sake of simplifying the discussion, assume first that the Markov chain is irreductible. (Whether it is the case or not depends on the protocol $\mathcal{P}$.) Let $Y_i(t)$ be the random variables equal to the numbers of agents in state q_i at time t, and let $Y(t) = (Y_1(t), \ldots, Y_{|Q|}(t))$. Let us now consider the Markov chain defined by $\overline{Y}(t) = \frac{1}{n}(Y_1(t), \cdots, Y_{|Q|}(t))$. A consequence of the Ergodic Theorem (this is where we use the irreducibility assumption) is that the chain $\overline{Y}(t)$ admits a unique stationary distribution, say $\mu = (\mu_1, \mu_2, \ldots, \mu_{|Q|})$. Hence, for any initial state of the population, the distribution of $\overline{Y}(t)$ converges to μ when t goes to infinity. In particular, the distribution of $\overline{Y}_1(t)$, the proportion of agents in the marked state q_1 at time t, converges to the distribution μ_1 when t goes to infinity. As a consequence, the expected value of $\overline{Y}_1(t)$ converges to the expected value $\mathbf{E}\mu_1$ of μ_1. Intuitively, we are interested in the limit of $\mathbf{E}\mu_1$ when n grows to infinity. The difficulty comes from the fact that a protocol is dealing with $\overline{Y}_1(t)$, which depends on both t and n. The study of this double limit must be treated with care in the general case, which is the purpose of the remainder of this section.

Notice that the limit of $\mathbf{E}\mu_1$ can be a non-rational real number, whereas, when n is fixed, the expected value of μ_1 is necessarily a rational number since, for every i, we have $\overline{Y}_i(t) \in \{\frac{1}{n}, \ldots, \frac{n}{n}\}$. So $\overline{Y}_i(t)$ is a Markov chain over this latter set. As a consequence, the distribution μ_i is a distribution over this latter set. Since the stationary distribution μ is the solution of a set of linear equations with rational coefficients, μ is necessarily weighting the elements of $\{\frac{1}{n}, \ldots, \frac{n}{n}\}$ with rational quantities. In particular, the expected value of μ_1 satisfies $\mathbf{E}\mu_1 = \sum_{i=1}^{n} \mu_1(\frac{i}{n}) \cdot \frac{i}{n}$, and thus is a rational number.

To handle the growth of the population, one must perform a time rescaling. Let us redefine the notations so that to capture explicitely the size n of the population. Let $Y_i^{(n)}(t)$ be the random variable equal to the numbers of agents in state q_i at time t in a population of size n, and let $\overline{Y}_i^{(n)}(t) = \frac{1}{n} \cdot Y_i^{(n)}(t)$. Let $\overline{Y}^{(n)}(t) = (\overline{Y}_1^{(n)}(t), \ldots, \overline{Y}_{|Q|}^{(n)}(t))$. Then let

$$X^{(n)}(t) = \overline{Y}^{(n)}(\lfloor nt \rfloor) + (nt - \lfloor nt \rfloor) \cdot (\overline{Y}^{(n)}(\lfloor nt + 1 \rfloor) - \overline{Y}^{(n)}(\lfloor nt \rfloor)).$$

By definition, $X^{(n)}(t)$ is a continuous-time Markov chain obtained by linear interpolation of $\overline{Y}^{(n)}$ with a time-acceleration of factor n. After rescaling, the number of interactions occurring in one time-unit is proportional to the number of agents in the population. To capture the asymptotic behavior of $X^{(n)}(t)$, we use a *balance* equation. Let $(e_q)_{q \in Q}$ be the canonical base of $\mathbb{R}^{|Q|}$, and let $b : \mathbb{R}^{|Q|} \to \mathbb{R}^{|Q|}$ be the function defined by:

$$b(x) = \sum_{(q_1,q_2) \in Q^2} \left(x_{q_1} x_{q_2} \Big(- e_{q_1} - e_{q_2} + \sum_{(q_3,q_4) \in Q^2} \delta_{q_1,q_2,q_3,q_4} (e_{q_3} + e_{q_4}) \Big) \right) \quad (3)$$

where $\delta_{q_1,q_2,q_3,q_4} = 1$ if $\Delta(q_1, q_2) = (q_3, q_4)$, and 0 otherwise. The function b acts as a balance equation. That is, assuming that the proportion of agents in state q is $x_q \in \mathbb{R}$ for every $q \in Q$, then one expects each rule $q_i\ q_j \to q_k\ q_l$ to happen with probability $x_{q_i} x_{q_j}$. Accounting for this balance for all rules, and considering that all produced states must be consumed by some rule, yield that if the proportion of states converges to some equilibrium x, then this equilibrium must satisfy $b(x) = 0$. The following has been proved in [14].

Lemma 1 (Theorem 4 of [14]). *For every initial condition $Y^{(n)}(0)$ with $Y^{(n)}(0) \to x$ when $n \to \infty$, the sequence of random processes $X^{(n)}$ converges in law to the solution of the stochastic differential equation (with degenerated brownian motion): $dX(t) = b(X(t))dt$, with $X(0) = x$.*

2.2 Computing with LPPs

In view of Lemma 1, we get that the behavior of a protocol can be well approached by an ordinary differential equation, when the size of the population becomes large. In particular, if x^* is some stable equilibrium of the differential equation, then one expects $\overline{Y}^{(n)}(t)$ to converge to x^* whenever it starts close enough to x^*. Unfortunately, the notion of convergence involved in Lemma 1 (i.e., convergence in law) is too weak to derive this conclusion directly. On the other hand, it is actually possible to go further in the analysis of population protocol, in order to provide a deeper understanding of the convergence. By doing so, we are able to provide an asymptotic development for $\overline{Y}^{(n)}(t)$, as stated in the following result.

Theorem 1. *Assume that the ordinary differential in Lemma 1 has a stable equilibrium $b(x^*) = 0$. Then there exists a neighborhood of x^* such that, whenever $\overline{Y}^{(n)}(0)$ belongs to this neighborhood, we have $\overline{Y}^{(n)}(t) \approx x^* + \frac{1}{\sqrt{n}} \mathcal{N}(0, \chi)$ when $t \to \infty$, where $\mathcal{N}(0, \chi)$ is the centered gaussian distribution with covariance matrix χ, for some χ, and $\approx$ denotes convergence in law of the rescaling of $\overline{Y}^{(n)}(t)$ when $t \to \infty$.*

By an adaptation of the arguments in [10], one can also show that if the ordinary differential equation in Lemma 1 has a stable equilibrium $b(x^*) = 0$, then, for every $\epsilon > 0$, and for every $0 < p < 1$, there is a neighborhood U of x^* and some integers n and t, both polynomial in $1/\epsilon$, which guarantee that, with probability at least p, we have $\|\overline{Y}^{(n)}(t) - x^*\| \leq \epsilon$ whenever the initial configuration belongs to U.

We have now all ingredients to formally define computing with LPPs.

Let $\mathcal{P} = (Q, \Delta)$ be a LPP. A vector of real numbers $x^* = (x_1, \ldots, x_{|Q|}) \in [0,1]^{|Q|}$ is said to be an *equilibrium* of a $\mathcal{P}$ if and only if $b(x^*) = 0$, that is to say the constant solution $f(t) = (x_1, \ldots, x_{|Q|})$ is a fix-point solution of the differential equation in Lemma 1. An equilibrium x^* of $\mathcal{P}$ is said to be *stable* if it is the (exponentially) stable equilibrium of the associated ordinary differential equation. In other words, there is a neighborhood U of the equilibrium x^* such that any trajectory starting from U converges exponentially fast to the equilibrium. This is equivalent to saying that the Eigenvalues of the Jacobian matrix of b in x^* has negative real parts [25].

Definition 1. *A real number ν is said to be computable by LPP if there exists a vector $x^* = (x_1, x_2, ..., x_k) \in [0,1]^k$ such that $\sum_{i=1}^{k} x_i = 1$, and a LPP $\mathcal{P}$, admitting finitely many equilibria, such that $(x_1, x_2, ..., x_k)$ is a stable equilibrium of $\mathcal{P}$ and $\sum_{q_i \in Q^+} x_i = \nu$ where Q^+ is the set of marked states for $\mathcal{P}$.*

Notice that the above definition requires the system to have finitely many equilibria. This assumption is mainly to avoid pathological cases, in particular the case of idle systems $q\ q' \to q\ q'$ for all q, q'. Indeed, in idle systems, all initial states are equilibria, and such a system would compute any real of $[0,1]$, depending on the initial configuration.

3 The Computational Power of LPPs

In this section, we establish our main result:

Theorem 2. *Every $\nu \in [0,1]$ is computable by a LPP if and only if it is algebraic.*

We first prove that there is an intrinsic limitation to the power of LPPs, namely not a single transcendental number can be computed by LPPs. Indeed, a direct consequence of arguments from model theory (mainly Tarski's effective procedure for quantifier elimination over real closed fields) allows us to prove the following lemma:

Lemma 2. *For every $\nu \in [0,1]$, if ν is computable by a LPP then ν is algebraic.*

The remaining part of the section is entirely dedicated to proving that every algebraic number is indeed computable by a LPP. The proof is constructive, meaning that we describe how to construct a LPP computing ν, for any given algebraic number $\nu \in [0,1]$. The construction of the protocol is made in four stages, corresponding to the following four subsections. The first stage consists in the design of LPPs computing rational numbers. The second stage consists in using the computation of rational numbers as a subroutine for the emulation of probabilistic transition rules. This stage will allow us to consider LPPs with transition rules of the form

$$q_i \; q_j \to \alpha_{i,j,k,l} \; q_k \; q_l$$

to be understood as: the interaction between two agents in respective states q_i and q_j results in the two agents moving to respective states q_k and q_l with probability $\alpha_{i,j,k,l}$. Then, the third stage of our proof is the construction of a (probabilistic) protocol $\mathcal{P}$ admitting ν as an equilibrium. We assume we are given a degree-δ polynomial $P \in \mathbb{Q}[X]$ with root ν. The protocol $\mathcal{P}$ is based on one specific choice for another degree-δ polynomial $P' \in \mathbb{Q}[X]$, and, essentially, satisfies that $(x_1, \ldots, x_\delta) \in [0,1]^\delta$ is an equilibrium of $\mathcal{P}$ if and only if (1) $P'(x_1) = 0$, (2) $x_i = x_1^i$ for every $1 \leq i < \delta$, and (3) $x_\delta = 1 - \sum_{i=1}^{\delta} x_i$. Finally, the fourth stage of the construction consists in proving that we can actually enforce this protocol $\mathcal{P}$ to be stable near the equilibrium.

3.1 Computing Rationals

Lemma 3. *Let $\nu \in [0,1]$ be a rational number. There exists a LPP computing ν.*

Proof. We first show that, for every integer $k \in \mathbb{N}$, there exists a protocol that, given any initial configuration, converges to the unique equilibrium $(\frac{1}{k}, \ldots, \frac{1}{k})$. For this purpose, consider the protocol $\mathcal{M}$ over states $Q_k = \{1, \ldots, k\}$ given by the following transition rules: $i \; j \to (i+1) \; (j+1)$ where, for $q \in Q_k$, $q+1$ stands for $(q \bmod k) + 1$. The dynamic system describing this protocol is

$$\frac{dx_i}{dt} = 2(x_{i-1} - x_i).$$

If $f : \mathbb{R} \to [0,1]^k$ is a solution of this differential system, then, considering $g(t) = \|f(t) - (\frac{1}{k}, \ldots, \frac{1}{k})\|^2$, where $\|x\|$ is the euclidian norm of vector x, we get

$$\frac{dg(t)}{dt} = 4\sum_{i=1}^{k} x_i(x_{i-1} - x_i).$$

A simple induction on $k \in \mathbb{N}$ enables to show that $\frac{dg(t)}{dt} \leq 0$ for every vector $x \in [0,1]^k$, and $\frac{dg(t)}{dt} = 0$ if and only if $x_1 = x_2 = \ldots = x_k$, thereby proving that f converges to $(\frac{1}{k}, \cdots, \frac{1}{k})$ when $t \to \infty$. This, in particular, guarantees that $(\frac{1}{k}, \ldots, \frac{1}{k})$ is the only stable equilibrium of $\mathcal{M}$.

Now, let $\nu = p/q \in \mathbb{Q}$. Computing ν is achieved by using $\mathcal{M}$ as above, with $k = q$, and setting marked states as the first p states of Q. □

3.2 Derandomization

We now prove that considering LPPs with probabilistic transition rules where the probabilities are rational does not change the computing power of LPPs. Note that this result has its own independent interest. It essentially says that the random choice of the agents involved in the transition provides enough randomness, and that there are no further benefits from using randomization in the transition rules. Nevertheless, using probabilistic LPPs, or PLPPs for short, considerably simplifies the construction of LPPs computing algebraic numbers.

We focus on PLPPs, where the transition rules are defined by

$$q_i \ q_j \rightarrow \alpha_{i,j,k,l} \ q_k \ q_l$$

where all $\alpha_{i,j,k,l}$ are rational numbers[1]. Recall that such probabilistic transition rules mean that an interacting pair of agents in respective states q_i and q_j will move to respective states q_k and q_l, with probability $\alpha_{i,j,k,l}$. Of course, for such rules to be well-defined, we assume that for every pair $(q_i, q_j) \in Q^2$, we have

- for every $(q_k, q_l) \in Q^2$, $\alpha_{i,j,k,l} \geq 0$, and
- $\sum_{(q_k,q_l)\in Q^2} \alpha_{i,j,k,l} = 1$.

Notice that all previous definitions and statements can be easily extended to PLPPs. In particular, Lemma 1 and Theorem 1 still hold, by replacing function b in Eq. 3 by

$$b(x) = \sum_{(q_1,q_2)\in Q^2} x_{q_1} x_{q_2} \Big(- e_{q_1} - e_{q_2} + \sum_{(q_3,q_4)\in Q^2} \alpha_{q_1,q_2,q_3,q_4} (e_{q_3} + e_{q_4}) \Big).$$

Lemma 4. *Let $\nu \in [0,1]$, and assume that there exists a probabilistic LPP computing ν, with rational probabilities. Then there exists a (deterministic) LPP computing ν.*

3.3 Constructing Equilibria

In view of the previous two subsections, one can freely use probabilistic LPPs, whenever the probabilities are rational, in order to compute any algebraic number $\nu \in [0,1]$. In this section, we will not yet produce a probabilistic LPPs computing an algebraic number ν, as we will ignore stability which is only discussed in the next section, and solely focus on constructing a protocol with ν as an equilibrium.

Lemma 5. *For every algebraic number $\nu \in [0,1]$, there exist $\delta \in \mathbb{N}$, $\lambda \in \mathbb{Q}$, and a protocol $\mathcal{P}$ such that $(\nu, \lambda\nu^2, \lambda^2\nu^3, \ldots, \lambda^{\delta-2}\nu^{\delta-1}, 1 - \sum_{i=1}^{\delta-1} \lambda^{i-1}\nu^i)$ is an equilibrium of $\mathcal{P}$.*

[1] In fact, our derandomization technique could be extended to the case in which the $\alpha_{i,j,k,l}$ are computable by a LPP. This would however overload the presentation, and the stronger assumption that $\alpha_{i,j,k,l} \in \mathbb{Q}$ is anyway sufficient for our purpose.

Proof. Let $\nu \in (0,1]$ be an algebraic number, and let $P(X) = \sum_{i=0}^{\delta} a_i X^i, P \in \mathbb{Q}[X]$, be a polynomial such that $P(\nu) = 0$, and $P(0) > 0$. We claim that there exist a rational number $\epsilon \neq 0$, and a protocol $\mathcal{P}_\epsilon$ with equilibrium $(\nu, \lambda\nu^2, \lambda^2\nu^3, \ldots$ $\ldots, \lambda^{\delta-2}\nu^{\delta-1}, 1 - \sum_{i=1}^{\delta-1} \lambda^{i-1}\nu^i)$ described by the following differential equations:

$$\begin{cases} dx_1 = \epsilon(a_0 + a_1 x_1 + \sum_{i=2}^{\delta-1} \frac{a_{i+1}}{\lambda^{i-1}} x_{i-1} x_1) \\ dx_i = \lambda x_1 x_{i-1} - x_i \ \text{ for every } i = 2, \ldots, \delta - 1 \\ dx_\delta = -\sum_{i=1}^{\delta-1} dx_i. \end{cases} \tag{4}$$

where λ is a rational number such that $\lambda > 0$, and $\sum_{i=1}^{\delta-1} \lambda^{i-1}\nu^i \leq 1$. To establish that claim, we explicitly construct a protocol $\mathcal{P}_\epsilon$ over set of states $Q = \{1, \ldots, \delta\}$ with 1 serving as our marked state. Fix $\lambda \in \mathbb{Q}$, $\lambda > 0$ small enough, so that $\sum_{i=1}^{\delta-1} \lambda^{i-1}\nu^i \leq 1$. Then let

$$M = \max\left(\{|\frac{a_{i+1}}{\lambda^{i-1}} + 2a_0 + a_1|, i \in \{2, ..., \delta - 1\}\} \cup \{|a_2 + a_0 + a_1|, |2a_0 + a_1|, a_0\}\right)$$

and fix $\epsilon \in \mathbb{Q}$, $0 < \epsilon < \frac{1}{M}(1 - \frac{\lambda}{2})$. We define the family $(\alpha_{i,j,k,l})_{1 \leq i,j,k,l \leq \delta}$, that yields the transition rules for the protocol $\mathcal{P}_\epsilon$ as follows:

$$\begin{cases} i = 1, j = 1 & \Longrightarrow & \alpha_{1,1,1,1} = \epsilon\, \frac{a_2 + a_1 + a_0}{2} + \frac{1}{2} \text{ and } \alpha_{1,1,2,2} = \frac{\lambda}{2} \\ i = 1, 1 < j < \delta - 1 & \Longrightarrow & \alpha_{1,j,1,1} = \epsilon\, \frac{\frac{a_{j+1}}{\lambda^{j-1}} + 2a_0 + a_1}{4} + \frac{1}{2} \text{ and } \alpha_{1,j,j+1,j+1} = \frac{\lambda}{4} \\ i = 1, j = \delta - 1 & \Longrightarrow & \alpha_{1,j,1,1} = \epsilon\, \frac{\frac{2a_\delta}{k^{\delta-2}} + 2a_0 + a_1}{4} + \frac{1}{2} \\ i = 1, j = \delta & \Longrightarrow & \alpha_{1,j,1,1} = \epsilon\, \frac{2a_0 + a_1}{4} + \frac{1}{2} \\ 1 < i < \delta - 1, j = 1 & \Longrightarrow & \alpha_{i,1,1,1} = \epsilon\, \frac{\frac{a_{i+1}}{\lambda^{i-1}} + 2a_0 + a_1}{4} + \frac{1}{2} \text{ and } \alpha_{i,1,i+1,i+1} = \frac{\lambda}{4} \\ i = \delta - 1, j = 1 & \Longrightarrow & \alpha_{i,1,1,1} = \epsilon\, \frac{\frac{2a_\delta}{k^{\delta-2}} + 2a_0 + a_1}{4} + \frac{1}{2} \\ i = \delta, j = 1 & \Longrightarrow & \alpha_{i,1,1,1} = \epsilon\, \frac{2a_0 + a_1}{4} + \frac{1}{2} \\ i > 1, j > 1 & \Longrightarrow & \alpha_{i,j,1,1} = \epsilon\, \frac{a_0}{2} \end{cases}$$

And, for all $(k,l) \neq (\delta,\delta)$ not considered above, we set $\alpha_{i,j,k,l} = 0$. Finally, if $(k,l) = (\delta,\delta)$, then $\alpha_{i,j,\delta,\delta} = 1 - \sum_{(k,l) \neq (\delta,\delta)} \alpha_{i,j,k,l}$.

By definition of M and ϵ, it follows that, for any pair (i,j), if $(k,l) \neq (\delta,\delta)$, then $0 \leq \alpha_{i,j,k,l} \leq 1$. Moreover, we have $0 \leq \sum_{(k,l) \neq (\delta,\delta)} \alpha_{i,j,k,l} \leq 1$. Thus, for every (i,j), $0 \leq \alpha_{i,j,\delta,\delta} \leq 1$. Therefore, the family $(\alpha_{i,j,k,l})$ properly defines a protocol $\mathcal{P}_\epsilon$. We now show that this protocol satisfies our needs. By construction, the dynamic of $\mathcal{P}_\epsilon$ is captured by the following system :

$$\forall k \in \{1, \ldots, \delta\},\ dx_k = \sum_{l=1}^{\delta} \sum_{i,j} (\alpha_{i,j,k,l} + \alpha_{i,j,l,k}) x_i x_j - x_k$$

which precisely yields Eq. 4. □

3.4 Enforcing Stability

Perhaps surprisingly, stability does not come for free, and the construction of the previous section is not sufficient to conclude. One needs to enforce stability.

For that purpose, the protocol of the previous section is modified in order to satisfy the stability criteria from the theory of dynamic systems. The following result completes the proof of Theorem 2.

Lemma 6. *For every algebraic number* $\nu \in [0,1]$*, there exist* $\delta \in \mathbb{N}$*,* $\lambda \in \mathbb{Q}$*, and a protocol* $\mathcal{P}$ *such that* $(\nu, \lambda\nu^2, \lambda^2\nu^3, \ldots, \lambda^{\delta-2}\nu^{\delta-1}, 1 - \sum_{i=1}^{\delta-1} \lambda^{i-1}\nu^i)$ *is a* stable *equilibrium of* $\mathcal{P}$*.*

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science 266(5187), 1021 (1994)
2. Angluin, D., Aspnes, J., Chan, M., Fischer, M.J., Jiang, H., Peralta, R.: Stably Computable Properties of Network Graphs. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 63–74. Springer, Heidelberg (2005)
3. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: PODC, pp. 290–299 (2004)
4. Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: PODC, pp. 292–299 (2006)
5. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing 20(4), 279–304 (2007)
6. Angluin, D., Aspnes, J., Fischer, M.J., Jiang, H.: Self-stabilizing Population Protocols. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 103–117. Springer, Heidelberg (2006)
7. Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. Distributed Computing 21(3), 183–199 (2008)
8. Angluin, D., Aspnes, J., Eisenstat, D.: A simple population protocol for fast robust approximate majority. Distributed Computing 21(2), 87–102 (2008)
9. Aspnes, J., Ruppert, E.: An introduction to population protocols. Bulletin of the EATCS 93, 106–125 (2007)
10. Aupy, G., Bournez, O.: On the number of binary-minded individuals required to compute $\sqrt{1/2}$. Theoretical Computer Science 412(22), 2219–2456 (2010)
11. Berry, G.: The chemical abstract machine. TCS 96(1), 217–248 (1992)
12. Blondel, V., Hendrickx, J., Olshevsky, A., Tsitsiklis, J.: Convergence in multiagent coordination, consensus, and flocking. In: 44th IEEE Conf. on Decision and Control, pp. 2996–3000 (2005)
13. Bournez, O., Chalopin, J., Cohen, J., Koegler, X., Rabie, M.: Computing with Pavlovian Populations. In: Fernàndez Anta, A., Lipari, G., Roy, M. (eds.) OPODIS 2011. LNCS, vol. 7109, pp. 409–420. Springer, Heidelberg (2011)
14. Bournez, O., Chassaing, P., Cohen, J., Gerin, L., Koegler, X.: On the convergence of population protocols when population goes to infinity. Applied Math. and Computation (2009)
15. Chaintreau, A., Hui, P., Crowcroft, J., Diot, C., Gass, R., Scott, J.: Impact of human mobility on opportunistic forwarding algorithms. IEEE Transactions on Mobile Computing 6(6), 606–620 (2007)
16. Chatzigiannakis, I., Michail, O., Spirakis, P.G.: Algorithmic Verification of Population Protocols. In: Dolev, S., Cobb, J., Fischer, M., Yung, M. (eds.) SSS 2010. LNCS, vol. 6366, pp. 221–235. Springer, Heidelberg (2010)

17. Chatzigiannakis, I., Spirakis, P.G.: The Dynamics of Probabilistic Population Protocols. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 498–499. Springer, Heidelberg (2008)
18. Chazelle, B.: Natural algorithms. In: SODA, pp. 422–431 (2009)
19. Clément, J., Delporte-Gallet, C., Fauconnier, H., Sighireanu, M.: Guidelines for the verification of population protocols. In: ICDCS, pp. 215–224 (2011)
20. Cucker, F., Smale, S.: Emergent behavior in flocks. IEEE Transactions on Automatic Control 52(5), 852–862 (2007)
21. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Ruppert, E.: When Birds Die: Making Population Protocols Fault-Tolerant. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 51–66. Springer, Heidelberg (2006)
22. Fernández, A., Gramoli, V., Jiménez, E., Kermarrec, A.-M., Raynal, M.: Distributed slicing in dynamic systems. In: ICDCS (2007)
23. Gramoli, V., Vigfusson, Y., Birman, K., Kermarrec, A.-M., van Renesse, R.: Slicing distributed systems. IEEE Trans. Computers 58(11), 1444–1455 (2009)
24. Gupta, I., Nagda, M., Devaraj, C.F.: The design of novel distributed protocols from differential equations. Distributed Computing 20(2), 95–114 (2007)
25. Hirsch, M.W., Smale, S., Devaney, R.: Differential Equations, Dynamical Systems, and an Introduction to Chaos. Elsevier Academic Press (2003)
26. Hofbauer, J., Sigmund, K.: Evolutionary game dynamics. Bulletin of the American Mathematical Society 4, 479–519 (2003)
27. Hui, P., Chaintreau, A., Scott, J., Gass, R., Crowcroft, J., Diot, C.: Pocket switched networks and human mobility in conference environments. In: WDTN, pp. 244–251 (2005)
28. Jelasity, M., Kermarrec, A.-M.: Ordered slicing of very large-scale overlay networks. In: Sixth IEEE International Conference on Peer-to-Peer Computing, P2P, pp. 117–124 (2006)
29. Juang, P., Oki, H., Wang, Y., Martonosi, M., Shiuan Peh, L., Rubenstein, D.: Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In: ASPLOS, pp. 96–107 (2002)
30. Michail, O., Chatzigiannakis, I., Spirakis, P.G.: Mediated population protocols. Theor. Comput. Sci. 412(22), 2434–2450 (2011)
31. Murray, J.D.: Mathematical Biology. I: An Introduction, 3rd edn. Springer (2002)
32. Wang, Y., Jain, S., Martonosi, M., Fall, K.: Erasure-coding based routing for opportunistic networks. In: WTDN, pp. 229–236 (2005)

Pancake Flipping Is Hard

Laurent Bulteau, Guillaume Fertin, and Irena Rusu

Laboratoire d'Informatique de Nantes-Atlantique (LINA), UMR CNRS 6241
Université de Nantes, 2 rue de la Houssinière, 44322 Nantes Cedex 3, France
{Laurent.Bulteau,Guillaume.Fertin,Irena.Rusu}@univ-nantes.fr

Abstract. Pancake Flipping is the problem of sorting a stack of pancakes of different sizes (that is, a permutation), when the only allowed operation is to insert a spatula anywhere in the stack and to flip the pancakes above it (that is, to perform a prefix reversal). In the burnt variant, one side of each pancake is marked as burnt, and it is required to finish with all pancakes having the burnt side down. Computing the optimal scenario for any stack of pancakes and determining the worst-case stack for any stack size have been challenges over more than three decades. Beyond being an intriguing combinatorial problem in itself, it also yields applications, e.g. in parallel computing and computational biology.

In this paper, we show that the Pancake Flipping problem, in its original (unburnt) variant, is NP-hard, thus answering the long-standing question of its computational complexity.

Keywords: Pancake problem, Computational complexity, Permutations, Prefix reversals.

1 Introduction

The pancake problem was stated in [7] as follows:

> The chef in our place is sloppy, and when he prepares a stack of pancakes they come out all different sizes. Therefore, when I deliver them to a customer, on the way to the table I rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom) by grabbing several from the top and flipping them over, repeating this (varying the number I flip) as many times as necessary. If there are n pancakes, what is the maximum number of flips (as a function of n) that I will ever have to use to rearrange them?

Stacks of pancakes are represented by permutations, and a flip consists in reversing a prefix of any length. The previous puzzle yields two entangled problems:

- Designing an algorithm that sorts any permutation with a minimum number of flips (this optimization problem is called MIN-SBPR, for Sorting By Prefix Reversals).
- Computing $f(n)$, the maximum number of flips required to sort a permutation of size n (the diameter of the so-called *pancake network*).

Gates and Papadimitriou [9] introduced the *burnt* variant of the problem: the pancakes are two-sided, and an additional constraint requires the pancakes to end with the unburnt side up. The diameter of the corresponding *burnt pancake network* is denoted $g(n)$. A number of studies [4–6, 9, 11–13] have aimed at determining more precisely the values of $f(n)$ and $g(n)$, with the following results:

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 247–258, 2012.

- $f(n)$ and $g(n)$ are known exactly for $n \leq 19$ and $n \leq 17$, respectively [5].
- $15n/14 \leq f(n) \leq 18n/11 + O(1)$ [12, 4].
- $\lfloor(3n+3)/2\rfloor \leq g(n) \leq 2n - 6$ [5] (upper bound for $n \geq 16$).

Considering MIN-SBPR, 2-approximation algorithms have been designed, both for the burnt and unburnt variants [6, 8]. Moreover, Labarre and Cibulka [13] have characterized a subclass of signed permutations, called *simple permutations*, that can be sorted in polynomial time.

The pancake problems have various applications. For instance, the pancake network, having both a small degree and diameter, is of interest in parallel computing. The algorithmic aspect, i.e. the sorting problem, has applications in comparative genomics, since prefix reversals are possible elementary modifications that can affect a genome during evolution. A related problem is Sorting By Reversals [1] where any subsequence can be flipped at any step, not only prefixes. This problem is now well-known, with a polynomial-time exact algorithm [10] for the signed case, and a 1.375-approximation [2] for the APX-hard unsigned case [3].

In this paper, we prove that the MIN-SBPR problem is NP-hard (in its unburnt variant), thus answering a question which has remained open for several decades. We in fact prove a stronger result: it is known that the number of breakpoints of a permutation (that is, the number of pairs of consecutive elements that are not consecutive in the identity permutation) is a lower bound on the number of flips necessary to sort a permutation. We show that deciding whether this bound is tight is already NP-hard.

2 Notations

We denote by $[\![a\,;\,b]\!]$ the interval $\{a, a+1, \ldots, b\}$ Let n be an integer. Input sequences are permutations of $[\![1\,;\,n]\!]$, hence we consider only sequences where all elements are unsigned, and there cannot be duplicates. We use upper case for sequences, and lower case for elements.

Consider a sequence S of length n, $S = \langle x_1, x_2, \ldots, x_n \rangle$. Element x_1 is said to be the *head element* of S. Sequence S has a *breakpoint* at position r, $1 \leq r < n$ if $x_r \neq x_{r+1} - 1$ and $x_r \neq x_{r+1} + 1$. It has a *breakpoint* at position n if $x_n \neq n$. We write $d_b(S)$ the number of breakpoints of S. Note that having $x_1 \neq 1$ does not directly count as a breakpoint, and that $d_b(S) \leq n$ for any sequence of length n. For any $p \leq q \in \mathbb{N}$, we write $\mathcal{I}_q^p$ the sequence $\langle p, p{+}1, p{+}2, \ldots, q \rangle$. $\mathcal{I}_n^1$ is the *identity*. For a sequence of any length $S = \langle x_1, x_2, \ldots, x_k \rangle$, we write $^\star S$ the sequence obtained by reversing S: $^\star S = \langle x_k, x_{k-1}, \ldots, x_1 \rangle$. Given an integer p, we write $p + S = \langle p + x_1, p + x_2, \ldots, p + x_k \rangle$.

The *flip* of length r is the operation that consists in reversing the r first elements of the sequence. It transforms

$$\begin{aligned} S &= \langle x_1, x_2, \ldots, x_r, \quad x_{r+1}, \ldots, x_n \rangle \\ \text{into} \quad S' &= \langle x_r, x_{r-1}, \ldots, x_1, \quad x_{r+1}, \ldots, x_n \rangle. \end{aligned}$$

Property 1. Given a sequence S' obtained from a sequence S by performing one flip, we have $d_b(S') - d_b(S) \in \{-1, 0, 1\}$.

$$\begin{tikzcd}
 & \langle 1,\ 3,\ 2,\ 5,\ 4\rangle \arrow[r] & \bot & & \\
\langle 5,\ 2,\ 3,\ \underline{1},\ \underline{4}\rangle \arrow[ur] \arrow[r] & \langle 4,\ \underline{1},\ 3,\ \underline{2},\ 5\rangle \arrow[r] \arrow[dr] & \langle 2,\ \underline{3},\ 1,\ 4,\ 5\rangle \arrow[r] & \langle 3,\ 2,\ \underline{1},\ 4,\ 5\rangle \arrow[r] & \langle 1,\ 2,\ 3,\ 4,\ 5\rangle \\
 & & \langle 1,\ 4,\ 3,\ 2,\ 5\rangle \arrow[r] & \bot & \\
\langle 5,\ 2,\ 3,\ 4,\ \underline{1}\rangle \arrow[r] & \langle 1,\ 4,\ 3,\ 2,\ 5\rangle \arrow[r] & \bot & &
\end{tikzcd}$$

Fig. 1. Examples of efficient flips. Sequence $\langle 5,\ 2,\ 3,\ 1,\ 4\rangle$ is efficiently sortable (in four flips), but $\langle 5,\ 2,\ 3,\ 4,\ 1\rangle$ is not.

A flip from S to S' is said to be *efficient* if $d_b(S') = d_b(S) - 1$, and we reserve the notation $S \to S'$ for such flips. A sequence of size n, different from the identity, is a *deadlock* if it yields no efficient flip, and we write $S \to \bot$. By convention, we underline in a sequence the positions corresponding to possible efficient flips: there are at most two of them, and at least one if the sequence is neither a deadlock nor the identity.

A *path* is a series of flips, it is *efficient* if each flip is efficient in the series. A sequence S is *efficiently sortable* if there exists an efficient path from S to the identity permutation (equivalently, if it can be sorted in $d_b(S)$ flips). See for example Figure 1.

Let S be a sequence different from the identity, and $\mathbb{T}$ be a set of sequences. We write $S \Longrightarrow \mathbb{T}$ if both following conditions are satisfied:

1. for each $T \in \mathbb{T}$, there exists an efficient path from S to T.
2. for each efficient path from S to the identity, there exists a sequence $T \in \mathbb{T}$ such that the path goes through T.

If $\mathbb{T}$ consists of a single element ($\mathbb{T} = \{T\}$), we may write $S \Longrightarrow T$ instead of $S \Longrightarrow \{T\}$. Note that condition 1. is trivial if $\mathbb{T} = \emptyset$, and condition 2. is trivial if there is no efficient path from S to $\mathcal{I}_n^1$. Given a sequence S, there can be several different sets $\mathbb{T}$ such that $S \Longrightarrow \mathbb{T}$. The following properties are easily deduced from the definition of $\Longrightarrow$.

Property 2. Given any sequence $S \neq \mathcal{I}_n^1$,

$$S \Longrightarrow \mathcal{I}_n^1 \Leftrightarrow S \text{ is efficiently sortable.}$$
$$S \Longrightarrow \emptyset \Leftrightarrow S \text{ is not efficiently sortable.}$$

Property 3. If $S \Longrightarrow \{S_1, S_2\}$, $S_1 \Longrightarrow \mathbb{T}_1$ and $S_2 \Longrightarrow \mathbb{T}_2$, then $S \Longrightarrow \mathbb{T}_1 \cup \mathbb{T}_2$.

3 Reduction from 3-SAT

The reduction uses a number of gadget sequences in order to simulate boolean variables and clauses with subsequences. They are organized in two levels (where level-1 gadgets are directly defined by sequences of integers, and level-2 gadgets are defined using a pattern of level-1 gadgets). For each gadget we define, we derive a property characterizing the efficient paths that can be followed if some part of the gadget appears at the head of a sequence. The proofs for all these properties follow the same pattern, with no obstacle appart from the increasing complexity of the sequences, and only the one for the Dock gadget is given in this extended abstract.

3.1 Level-1 Gadgets

Dock. The dock gadget is the simplest we define. Its only goal is to store sequences of the kind ${}^{\star}\mathcal{I}_q^{p+1}$ (with $p < q$) out of the head of the sequence, without "disturbing" any other part.

Definition 1. *Given two integers p and q with $p < q$, the* dock*for* ${}^{\star}\mathcal{I}_q^{p+1}$ *is the sequence Dock$(p, q) = D$, where*

$$D = \langle p-1,\, p,\, q+1,\, q+2 \rangle.$$

It has the following property:

Property 4. Let p and q be any integers with $p < q$, $D = Dock(p, q)$, and X and Y be any sequences. We have

$$\langle {}^{\star}\mathcal{I}_q^{p+1},\, X,\, D,\, Y \rangle \Longrightarrow \langle X,\, \mathcal{I}_{q+2}^{p-1},\, Y \rangle$$

Proof. An efficient path from $\langle {}^{\star}\mathcal{I}_q^{p+1},\, X,\, D,\, Y \rangle$ to $\langle X,\, \mathcal{I}_{q+2}^{p-1},\, Y \rangle$ is given by:

$$\begin{aligned} \langle {}^{\star}\mathcal{I}_q^{p+1},\, X,\, D,\, Y \rangle &= \langle q,\, q-1,\, \ldots,\, p+2,\, p+1,\, X,\, p-1,\, \underline{p},\, q+1,\, q+2,\, Y \rangle \\ &\to \langle p,\, p-1,\, \underline{{}^{\star}X},\, p+1,\, p+2,\, \ldots,\, q-1,\, q,\, q+1,\, q+2,\, Y \rangle \\ &\to \langle X,\, p-1,\, p,\, p+1,\, p+2,\, \ldots,\, q-1,\, q,\, q+1,\, q+2,\, Y \rangle \\ &= \langle X,\, \mathcal{I}_{q+2}^{p-1},\, Y \rangle \end{aligned}$$

For each sequence in the path, we apply the only possible efficient flip, hence every efficient path between $\langle {}^{\star}\mathcal{I}_q^{p+1},\, X,\, D,\, Y \rangle$ and $\mathcal{I}_n^1$ (if such a path exists) begins with these two flips, and goes through $\langle X,\, \mathcal{I}_{q+2}^{p-1},\, Y \rangle$.

Lock. A lock gadget contains three parts: a sequence which is the lock itself, a key element that "opens" the lock, and a test element that checks whether the lock is open.

Definition 2. *For any integer p, Lock(p) is defined by Lock$(p) = (key, test, L)$, where*

$$\begin{gathered} key = p + 10 \qquad\qquad test = p + 7 \\ L = p + \langle 1,\, 2,\, 9,\, 8,\, 5,\, 6,\, 4,\, 3,\, 11,\, 12 \rangle \end{gathered}$$

Given a lock $(key, test, L) = Lock(p)$, we write

$$L^o = p + \langle 1,\, 2,\, 3,\, 4,\, 6,\, 5,\, 8,\, 9,\, 10,\, 11,\, 12 \rangle.$$

Sequences L and L^o represent the lock when it is closed and open, respectively. If a sequence containing a closed lock has *key* for head element, then efficient flips put the lock in open position. If it has *test* for head element, then it is a deadlock if and only if the lock is closed.

Property 5. Let p be any integer, $(\mathit{key}, \mathit{test}, L) = \mathit{Lock}(p)$, and X and Y be any sequences. We have

$$\begin{aligned}
&\textbf{a.} && \langle \mathit{key},\, X,\, L,\, Y\rangle \Longrightarrow \langle X,\, L^o,\, Y\rangle \\
&\textbf{b.} && \langle \mathit{test},\, X,\, L^o,\, Y\rangle \Longrightarrow \langle X,\, \mathcal{I}_{p+12}^{p+1},\, Y\rangle \\
&\textbf{c.} && \langle \mathit{test},\, X,\, L,\, Y\rangle \rightarrow \bot
\end{aligned}$$

We use locks to emulate literals of a boolean formula: variables "hold the keys", and in a first time open the locks corresponding to true literals. Each clause holds three test elements, corresponding to its three literals, and the clause is true if the lock is open for at least one of the test elements.

Hook. A hook gadget contains four parts: two sequences used as delimiters, a *take* element that takes the interval between the delimiters and places it in head, and a *put* element that does the reverse operation. Thus, the sequence between the delimiters can be stored anywhere until it is called by *take*, and then can be stored back using *put*.

Definition 3. *For any integer p, $\mathit{Hook}(p)$ is defined by $\mathit{Hook}(p) = (\mathit{take}, \mathit{put}, G, H)$, where*

$$\begin{aligned}
\mathit{take} &= p + 10 & \mathit{put} &= p + 7 \\
G &= p + \langle 3,\, 4\rangle & H &= p + \langle 12,\, 11,\, 6,\, 5,\, 9,\, 8,\, 2,\, 1\rangle.
\end{aligned}$$

Given a hook $(\mathit{take}, \mathit{put}, G, H) = \mathit{Hook}(p)$, we write

$$\begin{aligned}
G' &= p + \langle 12,\, 11,\, 6,\, 5,\, 4,\, 3\rangle & H' &= p + \langle 10,\, 9,\, 8,\, 2,\, 1\rangle \\
G'' &= p + \langle 3,\, 4,\, 5,\, 6,\, 7\rangle & H'' &= p + \langle 12,\, 11,\, 10,\, 9,\, 8,\, 2,\, 1\rangle.
\end{aligned}$$

Property 6. Let p be an integer, $(\mathit{take}, \mathit{put}, G, H) = \mathit{Hook}(p)$, and X, Y and Z be any sequences. We have

$$\begin{aligned}
&\textbf{a.} && \langle \mathit{take},\, X,\, G,\, Y,\, H,\, Z\rangle \Longrightarrow \langle Y,\, G',\, {}^{\star}X,\, H',\, Z\rangle \\
&\textbf{b.} && \langle \mathit{put},\, X,\, G',\, {}^{\star}Y,\, H',\, Z\rangle \Longrightarrow \langle Y,\, G'',\, X,\, H'',\, Z\rangle \\
&\textbf{c.} && \langle G'',\, X,\, H'',\, Y\rangle \Longrightarrow \langle X,\, {}^{\star}\mathcal{I}_{p+12}^{p+1},\, Y\rangle
\end{aligned}$$

Fork. A fork gadget implements choices. It contains two parts delimiting a sequence X. Any efficient path encountering a fork gadget follows one of two tracks, where either X or ${}^{\star}X$ appears at the head of the sequence at some point. Sequence X would typically contain a series of triggers for various gadgets (*key*, *take*, etc.), so that X and ${}^{\star}X$ differ in the order in which the gadgets are triggered.

Definition 4. *For any integer p, $\mathit{Fork}(p)$ is defined by $\mathit{Fork}(p) = (E, F)$, where*

$$E = p + \langle 11,\, 8,\, 7,\, 3\rangle \quad F = p + \langle 10,\, 9,\, 6,\, 12,\, 13,\, 4,\, 5,\, 15,\, 14,\, 2,\, 1\rangle.$$

Given a fork $(E, F) = \mathit{Fork}(p)$, we write

$$\begin{aligned}
F^1 &= p + \langle 10,\, 9,\, 6,\, 7,\, 8,\, 11,\, 12,\, 13,\, 14,\, 15,\, 5,\, 4,\, 3,\, 2,\, 1\rangle \\
F^2 &= p + \langle 3,\, 7,\, 8,\, 11,\, 10,\, 9,\, 6,\, 12,\, 13,\, 4,\, 5,\, 15,\, 14,\, 2,\, 1\rangle
\end{aligned}$$

Property 7. Let p be an integer, $(E, F) = \mathit{Fork}(p)$, and X, Y be any sequences. We have

$$\begin{array}{ll}
\textbf{a.} & \langle E, X, F, Y\rangle \Longrightarrow \{\langle X, F^1, Y\rangle, \langle {}^{\star}X, F^2, Y\rangle\} \\
\textbf{b.} & \langle F^1, Y\rangle \Longrightarrow \langle {}^{\star}\mathcal{I}_{p+15}^{p+1}, Y\rangle \\
\textbf{c.} & \langle F^2, Y\rangle \Longrightarrow \langle {}^{\star}\mathcal{I}_{p+15}^{p+1}, Y\rangle
\end{array}$$

3.2 Level-2 Gadgets

Literals. The following gadget is used only once in the reduction. It contains the locks corresponding to all literals of the formula.

Definition 5. *Let p and m be two integers, Literals(p, m) is defined by*

$$\begin{array}{rl}
& \mathit{Literals}(p, m) = (\mathit{key}_1, \ldots, \mathit{key}_m, \mathit{test}_1, \ldots, \mathit{test}_m, \Lambda) \\
\textit{where} \quad \forall i \in [\![1\,;\,m]\!]\,, & (\mathit{key}_i, \mathit{test}_i, L_i) = \mathit{Lock}(p + 12(i-1)) \\
& \Lambda = \langle L_1, L_2, \ldots, L_m\rangle
\end{array}$$

Let O and I be two disjoint subsets of $[\![1\,;\,m]\!]$. We use Λ_I^O for the sequence obtained from Λ by replacing L_i by L_i^o for all $i \in O$ and by $\mathcal{I}_{p+12i}^{p+12i-11}$ for all $i \in I$.

Elements of O correspond to open locks in Λ_I^O, while elements of I correspond to open locks which have moreover been tested. Note that $\Lambda_\emptyset^\emptyset = \Lambda$, and that $\Lambda_{[\![1\,;\,m]\!]}^\emptyset = \mathcal{I}_{p+12m}^{p+1}$.

Property 8. Let p and m be two integers, O and I be two disjoint subsets of $[\![1\,;\,m]\!]$, $(\mathit{key}_1, \ldots, \mathit{key}_m, \mathit{test}_1, \ldots, \mathit{test}_m, \Lambda) = \mathit{Literals}(p, m)$, and X be any sequence. We have

$$\begin{array}{lll}
\textbf{a.} & \forall i \in [\![1\,;\,m]\!] - O - I, & \langle \mathit{key}_i, X, \Lambda_I^O\rangle \Longrightarrow \langle X, \Lambda_I^{O\cup\{i\}}\rangle \\
\textbf{b.} & \forall i \in O, & \langle \mathit{test}_i, X, \Lambda_I^O\rangle \Longrightarrow \langle X, \Lambda_{I\cup\{i\}}^{O-\{i\}}\rangle \\
\textbf{c.} & \forall i \in [\![1\,;\,m]\!] - O, & \langle \mathit{test}_i, X, \Lambda_I^O\rangle \to \bot
\end{array}$$

Variable. In the rest of this section, we assume that p_Λ and m are two fixed integers, and we define the gadget $(\mathit{key}_1, \ldots, \mathit{key}_m, \mathit{test}_1, \ldots, \mathit{test}_m, \Lambda) = \mathit{Literals}(p_\Lambda, m)$. Thus, we can use elements key_i and test_i for $i \in [\![1\,;\,m]\!]$, and sequences Λ_I^O for any disjoint subsets O and I of $[\![1\,;\,m]\!]$.

We now define a gadget simulating a boolean variable x_i. It holds two series of *key* elements: the ones with indices in P (resp. N) open the locks corresponding to literals of the form x_i (resp. $\neg x_i$). When the triggering element, ν, is brought to the head, a choice has to be made between P and N, and the locks associated with the chosen set (and only them) are opened.

Definition 6. *Let P, N be two disjoint subsets of $[\![1\,;\,m]\!]$ ($P = \{p_1, p_2, \dots, p_q\}$, $N = \{n_1, n_2, \dots, n_{q'}\}$) and p be an integer, Variable(P, N, p) is defined by*

$$\begin{aligned} &\textit{Variable}(P, N, p) = (\nu, V, D) \\ &\textit{where} \quad (\textit{take}, \textit{put}, G, H) = \textit{Hook}(p+2), \quad (E, F) = \textit{Fork}(p+14), \\ &\textit{in} \quad \nu = \textit{take} \\ &\qquad V = \langle G,\ E,\ \textit{key}_{p_1},\ \dots,\ \textit{key}_{p_q},\ \textit{put},\ \textit{key}_{n_1},\ \dots,\ \textit{key}_{n_{q'}},\ F,\ H \rangle \\ &\qquad D = \textit{Dock}(p+2, p+29) \end{aligned}$$

Given a variable gadget $(\nu, V, D) = $ Variable(P, N, p), we write

$$V^1 = \langle G'',\ \textit{key}_{n_1},\ \dots,\ \textit{key}_{n_{q'}},\ F^1,\ H'' \rangle$$
$$V^2 = \langle G'',\ \textit{key}_{p_q},\ \dots,\ \textit{key}_{p_1},\ F^2,\ H'' \rangle$$

where G'', H'', F^1, F^2, come from the definitions of Hook (Definition 3) and Fork (Definition 4).

The following property determines the possible behavior of a variable gadget.

Property 9. Let P, N be two disjoint subsets of $[\![1\,;\,m]\!]$, p be an integer, X and Y be two sequences, O, I be two disjoint subsets of $[\![1\,;\,m]\!]$, and $(\nu, V, D) =$ *Variable*(P, N, p). For sub-property (a.) we require that $(P \cup N) \cap (O \cup I) = \emptyset$, for (b.) that $N \cap (O \cup I) = \emptyset$, and for (c.) that $P \cap (O \cup I) = \emptyset$ (these conditions are in fact necessarily satisfied by construction since all sequences considered are permutations). We have

$$\begin{aligned} &\textbf{a.} \quad \langle \nu,\ X,\ V,\ Y,\ \Lambda_I^O \rangle \Longrightarrow \left\{ \begin{matrix} \langle X,\ V^1,\ Y,\ \Lambda_I^{O \cup P} \rangle, \\ \langle X,\ V^2,\ Y,\ \Lambda_I^{O \cup N} \rangle \end{matrix} \right\} \\ &\textbf{b.} \quad \langle V^1,\ X,\ D,\ Y,\ \Lambda_I^O \rangle \Longrightarrow \langle X,\ \mathcal{I}_{p+31}^{p+1},\ Y,\ \Lambda_I^{O \cup N} \rangle \\ &\textbf{c.} \quad \langle V^2,\ X,\ D,\ Y,\ \Lambda_I^O \rangle \Longrightarrow \langle X,\ \mathcal{I}_{p+31}^{p+1},\ Y,\ \Lambda_I^{O \cup P} \rangle \end{aligned}$$

Clause. The following gadget simulates a 3-clause in a boolean formula. It holds the *test* elements for three locks, corresponding to three literals. When the triggering element, γ, is at the head of a sequence, three distinct efficient paths may be followed. In each such path, one of the three locks is tested: in other words, any efficient path leading to the identity requires one of the locks to be open.

Definition 7. *Let $a, b, c \in [\![1\,;\,m]\!]$ be pairwise distinct integers and p be an integer, Clause(a, b, c, p) is defined by*

$$\begin{aligned} &\textit{Clause}(a, b, c, p) = (\gamma, \Gamma, \Delta) \\ &\textit{where} \quad (E_1, F_1) = \textit{Fork}(p+2), \quad (\textit{take}_1, \textit{put}_1, G_1, H_1) = \textit{Hook}(p+21), \\ &\qquad\quad (E_2, F_2) = \textit{Fork}(p+45), \quad (\textit{take}_2, \textit{put}_2, G_2, H_2) = \textit{Hook}(p+33), \\ &\textit{in} \quad \gamma = \textit{take}_1 \\ &\qquad \Gamma = \langle G_1,\ E_1,\ \textit{take}_2,\ \textit{put}_1,\ \textit{test}_c,\ F_1,\ G_2,\ E_2,\ \textit{test}_a,\ \textit{put}_2,\ \textit{test}_b,\ F_2,\ H_2,\ H_1 \rangle \\ &\qquad \Delta = \langle \textit{Dock}(p+2, p+17),\ \textit{Dock}(p+21, p+60) \rangle \end{aligned}$$

Given a clause gadget $(\gamma, \varGamma, \varDelta) = \mathit{Clause}(a, b, c, p)$*, we write*

$$\begin{aligned}
\varGamma^1 &= \left\langle G''_1,\ \mathit{test}_c,\ F^1_1,\ G''_2,\ \mathit{test}_b,\ F^1_2,\ H''_2,\ H''_1 \right\rangle \\
\varGamma^2 &= \left\langle G''_1,\ \mathit{test}_c,\ F^1_1,\ G''_2,\ \mathit{test}_a,\ F^2_2,\ H''_2,\ H''_1 \right\rangle \\
\varGamma^3 &= \left\langle G''_1,\ \mathit{take}_2,\ F^2_1,\ G_2,\ E_2,\ \mathit{test}_a,\ \mathit{put}_2,\ \mathit{test}_b,\ F_2,\ H_2,\ H''_1 \right\rangle
\end{aligned}$$

The following two properties determine the possible behavior of a clause gadget. The main point is that, starting from a sequence $\left\langle \gamma,\ X,\ \varGamma,\ Y,\ \varLambda^O_I \right\rangle$, there is one efficient path for each true literal in the clause (ie. each literal with index in O).

Property 10. Let X and Y be any sequences, and O, I be two disjoint subsets of $[\![1\,;\,m]\!]$. We have

$$\left\langle \gamma,\ X,\ \varGamma,\ Y,\ \varLambda^O_I \right\rangle \Longrightarrow \mathbb{T},$$

where $\mathbb{T}$ contains from 0 to 3 sequences, and is defined by:

$$\begin{aligned}
\left\langle X,\ \varGamma^1,\ Y,\ \varLambda^{O-\{a\}}_{I\cup\{a\}} \right\rangle &\in \mathbb{T} \text{ iff } a \in O \\
\left\langle X,\ \varGamma^2,\ Y,\ \varLambda^{O-\{b\}}_{I\cup\{b\}} \right\rangle &\in \mathbb{T} \text{ iff } b \in O \\
\left\langle X,\ \varGamma^3,\ Y,\ \varLambda^{O-\{c\}}_{I\cup\{c\}} \right\rangle &\in \mathbb{T} \text{ iff } c \in O
\end{aligned}$$

Property 11. Let Y and Z be any sequences, and O, I be two disjoint subsets of $[\![1\,;\,m]\!]$. We have

a. If $b, c \in O$, then $\left\langle \varGamma^1,\ Y,\ \varDelta,\ Z,\ \varLambda^O_I \right\rangle \Longrightarrow \left\langle Y,\ \mathcal{I}^{p+1}_{p+62},\ Z,\ \varLambda^{O-\{b,c\}}_{I\cup\{b,c\}} \right\rangle$

b. If $a, c \in O$, then $\left\langle \varGamma^2,\ Y,\ \varDelta,\ Z,\ \varLambda^O_I \right\rangle \Longrightarrow \left\langle Y,\ \mathcal{I}^{p+1}_{p+62},\ Z,\ \varLambda^{O-\{a,c\}}_{I\cup\{a,c\}} \right\rangle$

c. If $a, b \in O$, then $\left\langle \varGamma^3,\ Y,\ \varDelta,\ Z,\ \varLambda^O_I \right\rangle \Longrightarrow \left\langle Y,\ \mathcal{I}^{p+1}_{p+62},\ Z,\ \varLambda^{O-\{a,b\}}_{I\cup\{a,b\}} \right\rangle$

3.3 Reduction

Let ϕ be a boolean formula over l variables in conjunctive normal form, such that each clause contains exactly three literals. We write k the number of clauses, $m = 3k$ the total number of literals, and $\{\lambda_1, \ldots, \lambda_m\}$ the set of literals. Let $n = 31l + 62k + 12m$.

Definition 8. *We define the sequence* S_ϕ *as the permutation of* $[\![1\,;\,n]\!]$ *obtained by:*

$(\mathit{key}_1, \ldots, \mathit{key}_m, \mathit{test}_1, \ldots, \mathit{test}_m, \varLambda) = \mathit{Literals}(31l + 62k, m)$

$$\begin{aligned}
\forall i \in [\![1\,;\,l]\!],\quad & P_i = \{j \in [\![1\,;\,m]\!] \mid \lambda_j = x_i\} \\
& N_i = \{j \in [\![1\,;\,m]\!] \mid \lambda_j = \neg x_i\} \\
& (\nu_i, V_i, D_i) = \mathit{Variable}(P_i, N_i, 31(i-1)), \\
\forall i \in [\![1\,;\,k]\!],\quad & (a_i, b_i, c_i) = \text{ indices such that the } i\text{-th clause of } \phi \text{ is } \lambda_{a_i} \vee \lambda_{b_i} \vee \lambda_{c_i} \\
& (\gamma_i, \varGamma_i, \varDelta_i) = \mathit{Clause}(a_i, b_i, c_i, 31l + 62(i-1))
\end{aligned}$$

$$S_\phi = \left\langle \nu_1, \ldots, \nu_l, \gamma_1, \ldots, \gamma_k, V_1, \ldots, V_l, \varGamma_1, \ldots, \varGamma_k, D_1, \ldots, D_l, \varDelta_1, \ldots, \varDelta_k, \varLambda^\emptyset_\emptyset \right\rangle$$

Two things should be noted in this definition. First, elements *key*$_i$ and *test*$_i$ are used in the clause and variable gadgets, although they are not explicitly stated in the parameters (cf. Definitions 6 and 7). Second, one could assume that literals are sorted in the formula ($\phi = (\lambda_1 \vee \lambda_2 \vee \lambda_3) \wedge \dots$), so that $a_i = 3i - 2$, $b_i = 3i - 1$ and $c_i = 3i$, but it is not necessary since these values are not used in the following.

We now aim at proving Theorem 1 (p. 257), which states that S_ϕ is efficiently sortable if and only if the formula ϕ is satisfiable. Several preliminary lemmas are necessary, and the overall process is illustrated in Figure 2.

Variable Assignment.

Definition 9. *A* full assignment *is a partition* $\mathcal{P} = (T, F)$ *of* $[\![1\,;\,l]\!]$. *Using notations from Definition 8, we define the sequence* $S_\phi[\mathcal{P}]$ *by:*

$$\textit{For all } i \in [\![1\,;\,l]\!]\,, \quad V_i' = \begin{cases} V_i^1 \textit{ if } i \in T \\ V_i^2 \textit{ if } i \in F \end{cases}$$

$$O = \bigcup_{i \in T} P_i \cup \bigcup_{i \in F} N_i$$

$$S_\phi[\mathcal{P}] = \left\langle \gamma_1, \dots, \gamma_k, V_1', \dots, V_l', \varGamma_1, \dots, \varGamma_k, D_1, \dots, D_l, \varDelta_1, \dots, \varDelta_k, \varLambda_\emptyset^O \right\rangle$$

With the following lemma, we ensure that any sequence of efficient flips from S_ϕ begins with a full assignment of the boolean variables, and every possible assignment can be reached using only efficient flips.

Lemma 1

$$S_\phi \Longrightarrow \{S_\phi[\mathcal{P}] \mid \mathcal{P} \textit{ full assignment}\}$$

Going through Clauses. Now that each variable is assigned a boolean value, we need to verify with each clause that this assignment satisfies the formula ϕ. This is done by selecting, for each clause, a literal which is true, and testing the corresponding lock. As in Definition 8, for any $i \in [\![1\,;\,k]\!]$ we write (a_i, b_i, c_i) the indices such that the i-th clause of ϕ is $\lambda_{a_i} \vee \lambda_{b_i} \vee \lambda_{c_i}$ (thus, $a_i, b_i, c_i \in [\![1\,;\,m]\!]$).

Definition 10. *Let* $\mathcal{P}$ *be a full assignment. A* full selection σ *is a subset of* $[\![1\,;\,m]\!]$ *such that, for each* $i \in [\![1\,;\,k]\!]$, $|\{a_i, b_i, c_i\} \cap \sigma| = 1$ *(hence* $|\sigma| = k$*). A full selection* σ *and a full assignment* $\mathcal{P} = (T, F)$ *are* compatible*, if, for every* $i \in \sigma$*, literal* λ_i *is true according to assignment* $\mathcal{P}$ *(that is,* $\lambda_i = x_j$ *and* $j \in T$*, or* $\lambda_i = \neg x_j$ *and* $j \in F$*). Given a full selection* σ *and a full assignment* $\mathcal{P} = (T, F)$ *which are compatible, we define the sequence* $S_\phi[\mathcal{P}, \sigma]$ *by:*

$$\forall i \in [\![1\,;\,l]\!]\,, \quad V_i' = \begin{cases} V_i^1 \textit{ if } i \in T \\ V_i^2 \textit{ if } i \in F \end{cases} \qquad \forall i \in [\![1\,;\,k]\!]\,, \quad \varGamma_i' = \begin{cases} \varGamma_i^1 \textit{ if } a_i \in \sigma \\ \varGamma_i^2 \textit{ if } b_i \in \sigma \\ \varGamma_i^3 \textit{ if } c_i \in \sigma \end{cases}$$

$$O = \bigcup_{i \in T} P_i \cup \bigcup_{i \in F} N_i - \sigma \qquad\qquad I = \sigma$$

$$S_\phi[\mathcal{P}, \sigma] = \left\langle V_1', \dots, V_l', \varGamma_1', \dots, \varGamma_k', D_1, \dots, D_l, \varDelta_1, \dots, \varDelta_k, \varLambda_I^O \right\rangle$$

$$S_\phi = \left\langle \nu_1, \dots, \nu_l, \gamma_1, \dots, \gamma_k, V_1, \dots, V_l, \varGamma_1, \dots, \varGamma_k, D_1, \dots, D_l, \varDelta_1, \dots, \varDelta_k, \varLambda^{\emptyset}_{\emptyset} \right\rangle$$

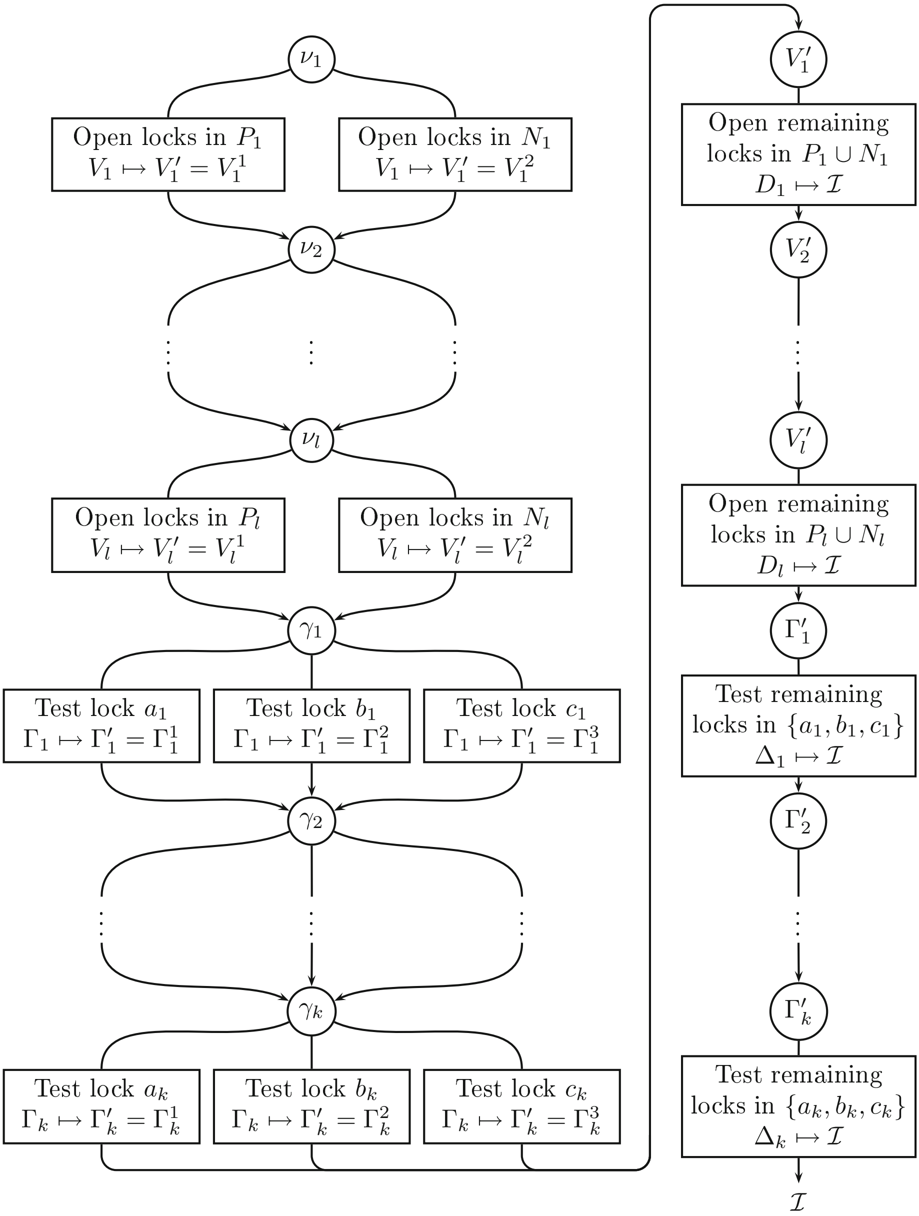

ν_1
Open locks in P_1 $V_1 \mapsto V_1' = V_1^1$
Open locks in N_1 $V_1 \mapsto V_1' = V_1^2$
ν_2
ν_l
Open locks in P_l $V_l \mapsto V_l' = V_l^1$
Open locks in N_l $V_l \mapsto V_l' = V_l^2$
γ_1
Test lock a_1 $\Gamma_1 \mapsto \Gamma_1' = \Gamma_1^1$
Test lock b_1 $\Gamma_1 \mapsto \Gamma_1' = \Gamma_1^2$
Test lock c_1 $\Gamma_1 \mapsto \Gamma_1' = \Gamma_1^3$
γ_2
γ_k
Test lock a_k $\Gamma_k \mapsto \Gamma_k' = \Gamma_k^1$
Test lock b_k $\Gamma_k \mapsto \Gamma_k' = \Gamma_k^2$
Test lock c_k $\Gamma_k \mapsto \Gamma_k' = \Gamma_k^3$
V_1'
Open remaining locks in $P_1 \cup N_1$ $D_1 \mapsto \mathcal{I}$
V_2'
V_l'
Open remaining locks in $P_l \cup N_l$ $D_l \mapsto \mathcal{I}$
Γ_1'
Test remaining locks in $\{a_1, b_1, c_1\}$ $\Delta_1 \mapsto \mathcal{I}$
Γ_2'
Γ_k'
Test remaining locks in $\{a_k, b_k, c_k\}$ $\Delta_k \mapsto \mathcal{I}$
$\mathcal{I}$

Fig. 2. Description of an efficient sorting of S_ϕ. Circular nodes correspond to head elements or sequences especially relevant (landmarks). We start with the head element of S_ϕ: ν_1. From each landmark, one, two or three paths are possible before reaching the next landmark, each path having its own effects, stated in rectangles, on the sequence. Possible effects are: transforming a subsequence of S_ϕ (symbol $\mapsto$), opening a lock, testing a lock (such a path requires the lock to be open).

With the following lemma, we ensure that after the truth assignment, every efficient path starting from S_ϕ needs to select a literal in each clause, under the constraint that the selection is compatible with the assignment.

Lemma 2. *Let $\mathcal{P}$ be a full assignment. Then*

$$S_\phi[\mathcal{P}] \Longrightarrow \{S_\phi[\mathcal{P}, \sigma] \mid \sigma \textit{ full selection compatible with } \mathcal{P}\}$$

Beyond Clauses

Lemma 3. *Let $\mathcal{P}$ be a full assignment and σ be a full selection, such that $\mathcal{P}$ and σ are compatible (provided such a pair exists for ϕ). Then*

$$S_\phi[\mathcal{P}, \sigma] \Longrightarrow \mathcal{I}_n^1$$

Theorem 1

$$S_\phi \Longrightarrow \mathcal{I}_n^1 \textit{ iff } \phi \textit{ is satisfiable.}$$

Proof. Assume first that $S_\phi \Longrightarrow \mathcal{I}_n^1$. By Lemma 1, there exists a full assignment $\mathcal{P} = (T, F)$ such that some path from S_ϕ to the identity uses $S_\phi[\mathcal{P}]$. Note that $S_\phi[\mathcal{P}] \Longrightarrow \mathcal{I}_n^1$. Now, by Lemma 2, there exists a full selection σ, compatible with $\mathcal{P}$, such that some path from $S_\phi[\mathcal{P}]$ to the identity uses $S_\phi[\mathcal{P}, \sigma]$. Consider the truth assignment $x_i :=$ True $\Leftrightarrow i \in T$. Then each clause of ϕ contains at least one literal that is true (the literal whose index is in σ), and thus ϕ is satisfiable.

Assume now that ϕ is satisfiable: consider any truth assignment making ϕ true, write T the set of indices such that $x_i =$ True, and $F = [\![1\,;\,l]\!] - T$. Write also σ a set containing, for each clause of ϕ, the index of one literal being true under this assignment. Then σ is a full selection, compatible with the full assignment $\mathcal{P} = (T, F)$. By Lemmas 1, 2 and 3 respectively, there exist efficient paths $S_\phi \Longrightarrow S_\phi[\mathcal{P}]$, $S_\phi[\mathcal{P}] \Longrightarrow S_\phi[\mathcal{P}, \sigma]$ and $S_\phi[\mathcal{P}, \sigma] \Longrightarrow \mathcal{I}_n^1$. Thus sequence S_ϕ is efficiently sortable.

Using Theorem 1, we can now prove the main result of the paper.

Theorem 2. *The following problems are* NP*-hard:*

- *Sorting By Prefix Reversals (MIN-SBPR)*
- *deciding, given a sequence S, whether S can be sorted in $d_b(S)$ flips*

Proof. By reduction from 3-SAT. Given any formula ϕ, create S_ϕ (see Definition 8, the construction requires a linear time). By Theorem 1, the minimum number of flips necessary to sort S_ϕ is $d_b(S_\phi)$ iff ϕ is satisfiable.

4 Conclusion

In this paper, we have shown that the Pancake Flipping problem is NP-hard, thus answering a long-standing open question. We have also provided a stronger result, namely, deciding whether a permutation can be sorted with no more than one flip per breakpoint is also NP-hard. However, the approximability of MIN-SBPR is still open: it can be seen that sequence S_ϕ can be sorted in $d_b(S_\phi) + 2$ flips, whatever the formula ϕ, hence this construction does not prove the APX-hardness of the problem.

Among related important problems, the last one having an open complexity is now the burnt variant of the Pancake Flipping problem. An interesting insight into this problem is given in a recent work from Labarre and Cibulka [13], where the authors characterize a subclass of permutations that can be sorted in polynomial time, using the breakpoint graph [1]. Another development consists in trying to improve the approximation ratio of 2 for the Pancake Flipping problem, both in its burnt and unburnt versions.

References

1. Bafna, V., Pevzner, P.: Genome rearrangements and sorting by reversals. In: FOCS, pp. 148–157. IEEE (1993)
2. Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-Approximation Algorithm for Sorting by Reversals. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 200–210. Springer, Heidelberg (2002)
3. Berman, P., Karpinski, M.: On Some Tighter Inapproximability Results (Extended Abstract). In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 200–209. Springer, Heidelberg (1999)
4. Chitturi, B., Fahle, W., Meng, Z., Morales, L., Shields, C.O., Sudborough, I., Voit, W.: An (18/11)n upper bound for sorting by prefix reversals. Theoretical Computer Science 410(36), 3372–3390 (2009)
5. Cibulka, J.: On average and highest number of flips in pancake sorting. Theoretical Computer Science 412(8-10), 822–834 (2011)
6. Cohen, D., Blum, M.: On the problem of sorting burnt pancakes. Discrete Applied Mathematics 61(2), 105–120 (1995)
7. Dweighter, H. American Mathematics Monthly, 82(1) (1975): (pseudonym of Goodman, J.E.)
8. Fischer, J., Ginzinger, S.: A 2-Approximation Algorithm for Sorting by Prefix Reversals. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 415–425. Springer, Heidelberg (2005)
9. Gates, W., Papadimitriou, C.: Bounds for sorting by prefix reversal. Discrete Mathematics 27(1), 47–57 (1979)
10. Hannenhalli, S., Pevzner, P.: Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. In: STOC, pp. 178–189. ACM (1995)
11. Heydari, M., Sudborough, I.: On Sorting by Prefix Reversals and the Diameter of Pancake Networks. In: Meyer auf der Heide, F., Rosenberg, A.L., Monien, B. (eds.) Heinz Nixdorf Symposium 1992. LNCS, vol. 678, pp. 218–227. Springer, Heidelberg (1993)
12. Heydari, M., Sudborough, I.: On the diameter of the pancake network. Journal of Algorithms 25(1), 67–94 (1997)
13. Labarre, A., Cibulka, J.: Polynomial-time sortable stacks of burnt pancakes. Theoretical Computer Science 412(8-10), 695–702 (2011)

In-place Heap Construction with Optimized Comparisons, Moves, and Cache Misses

Jingsen Chen[1], Stefan Edelkamp[2], Amr Elmasry[3], and Jyrki Katajainen[3]

[1] Department of Computer Science, Electrical and Space Engineering,
Luleå University of Technology, 971 87 Luleå, Sweden
[2] Faculty 3—Mathematics and Computer Science, University of Bremen,
P.O. Box 330 440, 28334 Bremen, Germany
[3] Department of Computer Science, University of Copenhagen,
Universitetsparken 1, 2100 Copenhagen East, Denmark

Abstract. We show how to build a binary heap in-place in linear time by performing $\sim 1.625n$ element comparisons, at most $\sim 2.125n$ element moves, and $\sim n/B$ cache misses, where n is the size of the input array, B the capacity of the cache line, and $\sim f(n)$ approaches $f(n)$ as n grows. The same bound for element comparisons was derived and conjectured to be optimal by Gonnet and Munro; however, their procedure requires $\Theta(n)$ pointers and does not have optimal cache behaviour. Our main idea is to mimic the Gonnet-Munro algorithm by converting a navigation pile into a binary heap. To construct a binary heap in-place, we use this algorithm to build bottom heaps of size $\Theta(\lg n)$ and adjust the heap order at the upper levels using Floyd's *sift-down* procedure. On another frontier, we compare different heap-construction alternatives in practice.

1 Introduction

The *binary heap*, introduced by Williams [16], is a binary tree in which each node stores one element. This tree is almost complete in the sense that all the levels are full, except perhaps the last level where elements are stored at the leftmost nodes. A binary heap is said to be *perfect* if it stores $2^k - 1$ elements, for a positive integer k. The elements are maintained in (min-)*heap order*, i.e. for each node the element stored at that node is not larger than the elements stored at its (at most) two children. A binary heap can be conveniently represented in an array where the elements are stored left-to-right in the level order of the tree.

In this paper we consider the problem of constructing a binary heap of n elements given in an array. Our objective is to do the construction in-place, i.e. using $O(1)$ words of additional memory. We assume that each word stores $O(\lg n)$ bits. The original algorithm of Williams constructs a binary heap in-place in $\Theta(n \lg n)$ time. Soon after, Floyd [6] improved the construction time to $\Theta(n)$ with at most $2n$ element comparisons. These classical results are covered by most textbooks on algorithms and data structures (see, e.g. [4, Chapter 6]).

In the literature, several performance indicators have been considered: the number of element comparisons, the number of element moves, and the number of

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 259–270, 2012.

Table 1. The number of element comparisons required by different heap-construction algorithms. The input is of size n; the average-case results assume that the input is a random permutation of n distinct elements.

Inventor	**Abbreviation**	**Worst case**	**Average case**	**Extra space**
Floyd [6]	**Alg. F**	$2n$	$\sim 1.88n$	$\Theta(1)$ words
Gonnet & Munro [8]	**Alg. GM**	$1.625n$	$1.625n$	$\Theta(n)$ words
McDiarmid & Reed [12]	**Alg. MR**	$2n$	$\sim 1.52n$	$\Theta(n)$ bits
Li & Reed [11]	**Lower bound**	$\sim 1.37n$	$\sim 1.37n$	$\Omega(1)$ words

cache misses. Even though Floyd's heap-construction algorithm is asymptotically optimal, for most performance indicators the exact optimal complexity bounds are still unknown. In Table 1 we summarize the best known bounds on the number of element comparisons when constructing a heap. By an element move we mean any assignment of elements to variables and array locations. Hence, a swap of two array elements is counted as three element moves and a cyclic rotation of k array elements is counted as $k+1$ element moves.

Without loss of generality, we assume that our heaps are perfect. As pointed out for example in [3], the nodes that do not root a perfect heap are located on the path from the last leaf to the root. This means that a heap of size n can be partitioned into at most $\lfloor \lg n \rfloor$ perfect subheaps. After building these subheaps, combining them can be done bottom-up in $O(\lg^2 n)$ time, and hence this does not affect the constant of the leading term in the complexity expressions.

Our main contribution is a simple technique for making existing algorithms for heap construction to run in-place. First, we reduce the amount of extra space used by the algorithm to a linear number of bits. Second, we use such an algorithm to build heaps of size $\Theta(\lg n)$ at the bottom of the input tree; we keep the needed bits in a constant number of words. Third, we combine these bottom heaps by exploiting the *sift-down* procedure of Floyd's algorithm at the top nodes of the tree. The key observation is that the work done by all *sift-down* calls is sublinear. We apply this approach for both the algorithm of Gonnet and Munro [8] and that of McDiarmid and Reed [12]. The inventors believe that their algorithms are optimal with respect to the number of element comparisons; the first in the worst-case sense and the second in the average-case sense.

In our in-place variant of the **GM** algorithm, we also optimize the number of element moves. For each bottom tree, we start by building a navigation pile at the end of the element array [9]; this technique is used by Kronrod [10]. To optimize the number of element moves when converting a navigation pile to a binary heap, we employ the hole technique that is also used by Floyd [6].

Another consequence of our in-place construction is that both algorithms can be modified—without affecting the number of element comparisons and element moves—such that the cache behaviour of the algorithms is almost optimal under reasonable assumptions. That is, without any knowledge about the size of cache blocks (B) and the size of fast memory (M), the algorithms incur about n/B cache misses. Here we rely on an improvement proposed by Bojesen et al. [1] showing how Floyd's algorithm can be made cache oblivious. For the algorithms involving linear extra space, this kind of optimality cannot be achieved due to the cache misses incurred when accessing the additional memory.

2 Heap Construction with a Navigation Pile

The **GM** algorithm [8] builds a binary heap on $2^k - 1$ elements in two phases: First a heap-ordered binomial tree [13] of size 2^k is constructed, and then this binomial tree is converted into a binary heap plus one *excess element* that is discarded. Since a binomial tree is a pointer-based data structure, we considered some related data structures that are more space-economical like a weak heap [5] and a navigation pile [9]. We decided to use the latter as our basic building block since it involves a fewer number of element moves for the overall construction.

2.1 Building a Navigation Pile

Consider an array of elements $a_0, \ldots, a_{n-1}$, where $n = 2^k$ for a positive integer k. A *navigation pile* [9] is a compact representation of a tournament tree built above this array. This tournament tree is a complete binary tree of size $2^k - 1$. Extending this tree by the element array, we get a complete binary tree of size $2^{k+1} - 1$. We say that the elements have height 0, all the bottommost nodes of the tournament tree have height 1, and so on; the root has height k.

Let us number the nodes at each level starting from 0. For a node at height h with index i at that level, its parent is at level $h + 1$ (if any) and has index $\lfloor i/2 \rfloor$, and its children are at level $h - 1$ (if any) and have indices $2i$ and $2i + 1$. These definitions are illustrated in Fig. 1.

A node of the tournament tree is said to *span* the leaves of the subtree rooted at that node. For each node we store a reference to the minimum element within the range spanned. If a node has height h, it spans 2^h elements. More precisely, if the index of the node is i at its level, it spans the elements in the range $[i \times 2^h \cdot\cdot\, i \times 2^h + 2^h - 1]$. To address any of these elements, we need to know the height of the node, its index, and an offset inside the range (these are h

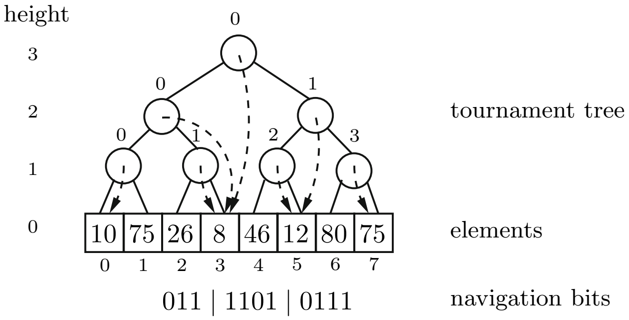


Fig. 1. A navigation pile of size 8; navigation information is visualized with pointers

bits called the *navigation bits*). The navigation information for the root uses k bits, that for the children of the root $k-1$ bits, and so on. The total number of navigation bits for all the nodes in the tournament tree is then:

$$2^k \sum_{h=1}^{k} h/2^h < 2n\,.$$

Hence, these bits can be stored in a bit-array of size at most $2n$ bits.

We can traverse the navigation pile starting from the root or from a leaf. During such a traversal we recall the height of the current node, its index at its present level, and the start position of the navigation bits of the present level in the bit array. When this information is available, we can access the parent or the children of the current node such that the same information is available for the accessed node. This can be accomplished by relying on simple arithmetic operations. For more details of implementing navigation piles and the related primitive operations, see [9].

To initialize a navigation pile, we traverse the tree bottom-up level by level and update the navigation information at each node by comparing the elements referred to by its children. Such a construction requires $n-1$ element comparisons and no element moves. In addition, a navigation pile supports the priority-queue operations *insert* and *delete* easily and efficiently [9].

2.2 Converting a Navigation Pile into a Binary Heap

Later we shall need a factory that can produce binary heaps of some particular size 2^k-1, where k is a positive integer. For the time being, let us assume that the factory already has a navigation pile on 2^k elements in its *input area* occupying the last locations of the element array. The task is to move the elements one by one from the input area to an *output area* that will, at the end, contain a binary heap of size 2^k-1. As another outcome of the procedure, we provide a reference to the location of the excess element in the input area.

For a tournament-tree node in a navigation pile, the first bit of its navigation information tells whether the minimum element among the elements it spans is in the left or the right subtree of that node. We call the subtree in which the minimum element is the *winner subtree*, and the other subtree the *loser subtree.*

As for the **GM** algorithm, our procedure is recursive. The nodes of the navigation pile are visited starting from the root, which is initially the *current input node.* Correspondingly, in the output area, the root of the binary heap to-be-built is initially the *current output node.* If the current input node is visited for the first time, we immediately proceed to its loser subtree and recursively convert it into a binary heap. The right child of the current output node will root the created binary heap. After processing the loser subtree, we move the minimum element w among those spanned by the current input node to the current output node, and then we move the excess element of the loser subtree to the old position of w. Hereafter, we have to update the navigation information on the path from w's old place to the root of the winner subtree. (It is not necessary to update the navigation information at the current input node.) It takes one element comparison per internal node to fix this navigation information. If the current input node is at height k of the navigation pile, we perform $k-1$ element comparisons. (This is similar to an *insert* operation in a navigation pile.) After this fix, the winner subtree can be converted into a binary heap recursively. The output is placed at the left subtree of the current output node.

As the base case, we use $k = 3$ (a navigation pile with 8 elements). The minimum element w of the structure is referred to by the root of the navigation pile and is readily known. Notice also that the loser subtree of the root forms a heap of size 3 plus one excess element. In other words, the minimum element among the elements of the loser subtree is readily known as well. We are only left with determining the minimum element within the winner subtree. In addition to w, there are three other elements in the winner subtree, and the smaller of two of them is already known. It follows that we only need to compare the third element with the smaller of these two elements. Hence, only one element comparison is needed for the base case.

Let us now analyse the performance of this conversion procedure. To convert a navigation pile of size 2^k into a binary heap, we perform two recursive calls for navigation piles of size 2^{k-1}. In addition, we need to call *insert* once to refresh the navigation information in the winner subtree. Let $C(2^k)$ be the number of element comparisons needed to convert a navigation pile of size 2^k into a binary heap plus an excess element. The number of element comparisons performed by *insert* is $k-1$. The next recursive relation follows:

$$\begin{cases} C(8) &= 1, \\ C(2^k) &= 2C(2^{k-1}) + k - 1\,. \end{cases}$$

For $n = 2^k \geq 8$, the solution of this relation is $C(n) = 5/8 \cdot n - \lg n - 1$. Adding the $n-1$ element comparisons needed for the construction of the navigation pile, the total number of element comparisons to build a binary heap on n elements is bounded by $1.625n$.

Let $M(2^k)$ be the number of element moves performed when converting a navigation pile of size 2^k into a binary heap. In the base case, every element except the excess element is moved from the input area to the output area. In the general case, after each recursive call, the minimum element is moved to the output area and the excess element of the loser subtree is moved to the place of the minimum element in the winner subtree. The next recursive relation follows:

$$\begin{cases} M(8) &= 7, \\ M(2^k) &= 2M(2^{k-1}) + 2\,. \end{cases}$$

For $n = 2^k \geq 8$, the solution of this relation is $M(n) = 9/8 \cdot n - 2$, i.e. the number of element moves in this case is bounded by $1.125n$.

Before proceeding, we have to consider one important detail. Since we aim at using this conversion procedure as a subroutine in our in-place algorithm, we must ensure that the recursion is handled only using a constant amount of extra storage. In an iterative implementation of the procedure, we keep track of the current input node (plus all the normal information required when traversing a navigation pile, including the present height), the current output node, and the previous input node. When this information is available, we can deduce whether the current input node is visited for the first, the second, or the third time; and whether we are processing a loser subtree or a winner subtree. So, our iterative implementation does not need any recursion stack.

3 Building a Binary Heap In-place

Assume that the task is to build a binary heap on n elements; here we need not make any assumptions about n. Let $m = 2^{\lfloor \lg \lg n \rfloor + 1} - 1$. We call all complete binary trees of size m *bottom trees*. The basic idea is to use the algorithm described in the previous section to convert all bottom trees to *bottom heaps* and then ensure the heap order at the upper levels by using the *sift-down* procedure of Floyd's algorithm or any of its variants. The crucial observation is that, by carefully choosing the sizes of the bottom trees, the work done by the *sift-down* calls at the upper levels becomes sublinear. The details are explained next.

To be able to convert the bottom trees into binary heaps, we use the last $m + 1$ locations of the element array as the input area for our factory. The main distinction from our earlier construction is that we do not have an empty output area, and hence we have to use the other bottom trees as the output area. Observe that there is a constant number of (at most three) bottom trees that overlap the input area. These *special bottom trees* will be handled differently.

In the first phase, we process all the bottom trees that are not special. Consider one such bottom tree T. We construct a navigation pile in the input area of our factory and convert it into a bottom heap as explained earlier. For that, the bottom tree T is used as the output area. Each time before an element is moved from the input area to T, the element at that particular location in T is moved to the input area. After this process, the input area is again full of elements and can be used for the construction of the next bottom heap.

In the second phase, we process the special bottom trees by using Floyd's heap-construction algorithm. Since the number of the special bottom trees is a constant, this phase only requires $O(\lg n)$ work.

In the third phase, we ensure the heap order for the nodes at the upper levels by using the *sift-down* procedure of Floyd's algorithm; this is done level by level starting at level $\lfloor \lg \lg n \rfloor + 1$ upwards. For each such node we compare the values of its two children, then compare its value x with the smaller of its two children, and move this child to the parent if the value at the child is smaller than x. If a move took place, we start from the moved child and repeat until we either reach a leaf or until x is not larger than both children. We finally move x to the vacant node if any moves were performed. Other variants of the *sift-down* procedure that require less element comparisons can be used, but this will not affect our overall bounds as the work performed in the third phase is sublinear.

Note that this construction is fully in-place. Since the size of the navigation pile is only logarithmic, the bits needed can be kept in a constant number of words. In addition to these bits, the first phase only requires a constant number of other information. Since Floyd's algorithm is in-place, the second and third phases also require a constant number of words.

To optimize the number of element moves, we make two modifications to the first phase. Let us call all complete binary trees of size 7 *micro trees*. In the base case, we have to move the corresponding elements from the navigation pile to a micro tree, and vice versa. An element swap would involve three element moves. To reduce this to two element moves, we put the elements of the first micro tree aside at the beginning of the procedure. Then we move the corresponding elements from the input area to this empty space once those elements are processed, and then fill these vacated positions of the input area with the elements from the next micro tree. These actions are repeated whenever a base case is to be processed. Higher up we can use the hole technique to reduce the number of element moves by one per element. In a non-optimized form, for every step of the algorithm, we should perform a cyclic rotation of three elements: the minimum element w referred to by the current input node of the navigation pile, the excess element of its loser subtree, and the element at the current output node; this would mean four element moves. However, the pattern how elements are moved to the output area is completely predictable. Therefore, we can move the parent of the root of the first micro tree (the first output node) aside at the beginning of the procedure. Thereafter, we can move the current minimum element w from the navigation pile to this hole, move the excess element to the position of w, and create a new hole by moving the element at the next output position to the place of the excess element. Repeating these actions, a cyclic rotation of three elements would involve three element moves. After processing all the bottom trees (except the special ones), the elements that were put aside are taken back to the current holes in the input area.

Let us now analyse the performance of this procedure. The number of elements involved in all the constructions of the bottom heaps is bounded by n. It follows that, if we use our version of the **GM** algorithm in the first phase, the number of element comparisons is bounded by $1.625n + o(n)$. Compared to our earlier construction, one more element move is needed for moving each element to the input area. Including the cost of putting elements aside, the number of element moves is bounded by $2.125n + o(n)$. In the second phase the amount of work done is $O(\lg n)$. In the third phase, the number of element comparisons and moves performed by the *sift-down* routine starting from a node at height h is at most $2h$. Since there are at most $n/2^{h+1}$ nodes at height h, and as we process the nodes at height $\lfloor \lg \lg n \rfloor + 1$ upwards, the total work (element comparisons and moves) performed in the third phase is proportional to at most

$$\sum_{h=\lfloor \lg \lg n \rfloor + 1}^{\lfloor \lg n \rfloor} 2h \cdot n/2^{h+1} = O\left(n \cdot \frac{\lg \lg n}{\lg n}\right) = o(n)\,.$$

We note that in the base case the element moves are handled more efficiently than at the upper levels of the tree. By making the base case larger one could get the number of element moves even closer to $2n$. With extra space for $\sim\lg n$ elements, the bound $\sim 2n$ element moves could actually be reached.

4 Improving the Cache Behaviour

Consider the construction of a heap of size n on a computer that has a two-level memory: a slow memory containing the elements of the input array and a fast memory, call it a *cache*, where the data must be present before it can be moved further to the register file of the computer. Assume that the size of the cache is M and that the data is transferred in blocks of size B between the two memory levels; both M and B are measured in elements. We assume that the cache is ideal, so the optimal off-line algorithm is the underlying block-replacement strategy when the cache is full. The ideal cache model is standard in the analysis of cache-oblivious algorithms [7].

For our analysis, we assume that $M \gg B\lg(\max\{M,n\})$. Under this assumption, several small heaps of size $\sim\lg n$ can simultaneously be inside the fast memory. When a heap is inside the fast memory, among the blocks containing the elements of the heap, there may be at most two blocks per heap level that also contain elements from outside the heap. By the assumption, a heap of size cM together with these $2\lceil\lg(cM)\rceil$ blocks, constituting $cM + 2B\lceil\lg(cM)\rceil$ elements, can simultaneously be inside the fast memory, provided that $c < 1$ is a small enough positive constant.

Consider an algorithm **A** (either our version of the **GM** algorithm or the **MR** algorithm) that is used to construct all bottom heaps of size $\sim\lg n$. All the remaining nodes are made part of the final heap by calling Floyd's *sift-down* routine. To improve the cache behaviour of the algorithm, the enhancement proposed by Bojesen et al. [1] is to handle these nodes in reverse depth-first order instead of reverse breadth-first order. This algorithm can be coded using only a constant amount of extra memory by recalling the level where we are at, the current node, and the node visited just before the current node. When n is a power of two minus one, the procedure is pretty simple. In the iterative version given in Fig. 2, the nodes at level $\lfloor\lg\lg n\rfloor$ are visited from right to left. After constructing a binary heap below such a node using algorithm **A**, its ancestors are visited one by one until an ancestor is met that is a right child (its index is odd); for each of the visited ancestors the *sift-down* routine is called.

Here we ignored the detail that our version of algorithm **A** uses the last part of the element array as its input area, but it is easy to sift down the elements of the special bottom trees and their ancestors in a separate post-processing phase. Namely, the nodes on the right spine going down from the root of the heap to the rightmost leaf are easy to detect (their indices are powers of two minus one); so, in the other phases, visiting these nodes can be avoided.

When processing a heap of size cM during the depth-first traversal, each block is read into fast memory only once. When such a heap has been processed, the blocks of the fast memory can be replaced arbitrarily, except that the blocks containing elements from outside this particular part are kept inside fast memory

```
in-place-A::make-heap(a: array of n elements, less: comparison function)
assert $n = 2^{\lceil \lg n \rceil} - 1$
$h \leftarrow \lfloor \lg \lg n \rfloor$   // height of the bottom trees
$j \leftarrow n/2^h$        // index of the root of the last bottom tree
$i \leftarrow j/2$          // index of the parent of the node with index $j$
while $j > i$
    A::make-heap($a$, $j$, $h$, less)
    $z \leftarrow j$
    while ($z$ bitand 1) = 0
        F::sift-down($a$, $z/2$, $n$, less)
        $z \leftarrow z/2$
    $j \leftarrow j - 1$
```

Fig. 2. In-place heap construction by traversing the nodes above the bottom trees in depth-first order. The root has index 1, and the leaves have height 0.

until their elements are processed. For the topmost $\sim n/(cM)$ elements, we can assume that each *sift-down* call incurs at most $\sim \lg n$ cache misses. Thus, the total number of cache misses incurred is at most $n/B + O(n \lg n/M)$. By our assumption, the first term in this formula is dominating.

5 Heap Construction in Practice

The heap-construction algorithm of Gonnet and Munro [8] is considered by many to be a theoretical achievement that has little practical significance. Out of curiosity, we wanted to investigate whether this belief is true or not; in particular, whether the improvements presented in this paper affect the state of affairs. Therefore, we implemented relaxed variants (with respect to the memory usage and the number of element moves performed) of the proposed algorithms and compared their performance to several existing algorithms. In this section we report the results of our experiments. All the implemented programs had the same interface as the C++ standard-library function `make_heap`.

The tests were performed on a 32-bit computer (model Intel® Core$^{\mathrm{TM}}$2 CPU T5600 @ 1.83GHz) running under Ubuntu 11.10 (Linux kernel 3.0.0-13-generic) using `g++` compiler (`gcc` version 4.6.1) with optimization level `-O3`. The size of L2 cache of this computer was about 2 MB and that of the main memory 1 GB.[1]

In an early stage of this study, we collected programs from public software repositories and wrote a number of new competitors for heap construction. In total, we looked at over 20 heap-construction methods including: Williams' algorithm of repeated insertions [16]; Floyd's algorithm of repeated merging with

[1] We also ran our experiments on two other 64-bit computers, one having an Intel i/7 CPU 2.67 GHz processor (Ubuntu 10.10 with Linux kernel 2.6.28-11-generic installed) and another having an AMD Phenom II X4 925 processor (4096 MB RAM and Linux Mint 11.0 installed).

top-down [6], bottom-up [14] and binary-search [2] *sift-down* policies, as well as depth-first and layered versions of it [1]; and McDiarmid and Reed's variant [12] that has the best known average-case performance.

We found that sifting down with binary search and explicitly maintaining a search path were inferior, so we excluded them from later rounds. The layered construction [1] that iteratively finds medians to build a heap bottom-up was fast on large inputs, but the number of element comparisons performed was high (larger than $2n$), so we also excluded it. Our implementation of Floyd's algorithm with the bottom-up *sift-down* policy had the same number of element comparisons as the built-in function in the C++ standard library and its running time was about the same, so we also excluded it. We looked at other engineered variants that include many implementation refinements, e.g. the code-tuned refinements discussed in [1], but they were non-effective, so we relied on Floyd's original implementation. The **GM** versions using weak heaps instead of navigation piles were consistently slower and performed more element moves, and hence were also excluded from this report.

The preliminary study thus left us with the following noteworthy competitors:

std: The implementation of the `make_heap` function that came with our `g++` compiler relying on the bottom-up *sift-down* policy [14]. Two of the underlying subroutines passed elements by value. Although this may result in unnecessary element moves, we assumed that these function calls will not involve any element moves.

F: We converted Floyd's original Algol program into C++. This program rotates the elements on the *sift-down* path cyclically, so element swaps are not used.

BKS: This variant of Floyd's heap-construction program traverses the nodes in depth-first order as discussed in [1].

in-situ GM: We implemented the algorithm of Gonnet and Munro [8] using a navigation pile as described in this paper, except that the subroutine for converting a navigation pile into a binary heap was recursive. Instead of bit-compaction techniques we used full indices at the nodes, and we did not use the hole technique for optimizing the number of element moves. Also, we had to make the height of the bottom trees $2\lfloor \lg\lg n \rfloor$ (instead of $\lfloor \lg\lg n \rfloor$) before the performance characteristics of the **GM** algorithm became visible.

in-situ MR: As in the previous program, the nodes were visited in depth-first order, but now the bottom trees were processed using the algorithm of McDiarmid and Reed [12]. All the elements on the *sift-down* path were moved cyclically first after the final position of the new element was known as proposed in [15]. Again, no bit-compaction was in use and full bytes were used instead of bits. The height of the bottom trees was also set to $2\lfloor \lg\lg n \rfloor$.

We tested the programs for random permutations of n distinct integers for different (small, medium, large, and very large) problem sizes $n = 2^{10} - 1$, $2^{15} - 1$, $2^{20} - 1$, and $2^{25} - 1$. The elements were of type `int`. All our programs were tuned to construct binary heaps of size $2^k - 1$. The obtained results are given in Fig. 3 (execution time), Fig. 4 (element comparisons), and Fig. 5 (element moves).

Program	std	F	BKS	in-situ	in-situ
n				GM	MR
$2^{10} - 1$	22.3	14.6	17.1	21.3	26.2
$2^{15} - 1$	22.2	14.6	17.4	23.0	24.4
$2^{20} - 1$	29.3	21.9	17.8	22.9	23.6
$2^{25} - 1$	29.8	21.7	17.5	22.9	23.6

Fig. 3. Execution time per element in nanoseconds

Program	std	F	BKS	in-situ	in-situ
n				GM	MR
$2^{10} - 1$	1.64	1.86	1.86	1.74	1.52
$2^{15} - 1$	1.64	1.88	1.88	1.65	1.54
$2^{20} - 1$	1.64	1.88	1.88	1.63	1.53
$2^{25} - 1$	1.65	1.88	1.88	1.63	1.53

Fig. 4. Number of element comparisons performed per element

Program	std	F	BKS	in-situ	in-situ
n				GM	MR
$2^{10} - 1$	2.25	1.73	1.73	2.06	1.52
$2^{15} - 1$	2.25	1.74	1.74	2.38	1.53
$2^{20} - 1$	2.25	1.74	1.74	2.38	1.53
$2^{25} - 1$	2.25	1.74	1.74	2.38	1.52

Fig. 5. Number of element moves performed per element

In our opinion—while not being overly tuned—our in-situ treatment for the **GM** algorithm showed acceptable practical performance. As expected, the number of element comparisons and element moves for the **in-situ GM** algorithm were beaten by the **in-situ MR** algorithm. Concerning the running time, the **in-situ GM** algorithm could beat the **in-situ MR** algorithm, but it was beaten by Floyd's algorithm **F** and the depth-first variant **BKS**.

6 Concluding Remarks

In practical terms, Floyd [6] solved the heap-construction problem in 1964. For our experiments we took his Algol program and converted it into C++ with very few modifications. For integer data, the cache-optimized version of Floyd's program described by Bojesen et al. [1] was the only program that could outperform Floyd's original program, and this happened only for large problem instances. The algorithms discussed in this paper can only outperform Floyd's program when element comparisons are expensive. In the worst case, Floyd's algorithm performs at most $2n$ element comparisons and $2n$ element moves. Furthermore, as shown in [1], by a simple modification, Floyd's algorithm can be made cache oblivious so that its cache behaviour is almost optimal.

In theoretical terms, the heap-construction problem remains fascinating. We showed how the two algorithms believed to be the best possible with respect to the number of element comparisons can be optimized with respect to the amount of space used and the number of cache misses incurred. In the worst case, our in-place variant of Gonnet and Munro's algorithm [8] requires $\sim 1.625n$ element comparisons and $\sim 2.125n$ element moves. We also showed that the same technique can be used to make McDiarmid and Reed's algorithm [12] in-place. Moreover, we proved that both of these in-place algorithms can be modified to demonstrate almost optimal cache behaviour.

The main question that is still not answered is: Can the bounds for heap construction be improved for any of the performance indicators considered?

References

1. Bojesen, J., Katajainen, J., Spork, M.: Performance engineering case study: Heap construction. ACM J. Exp. Algorithmics 5, Article 15 (2000)
2. Carlsson, S.: A variant of heapsort with almost optimal number of comparisons. Inform. Process. Lett. 24(4), 247–250 (1987)
3. Chen, J.: A Framework for Constructing Heap-Like Structures in-Place. In: Ng, K.W., Balasubramanian, N.V., Raghavan, P., Chin, F.Y.L. (eds.) ISAAC 1993. LNCS, vol. 762, pp. 118–127. Springer, Heidelberg (1993)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press, Cambridge (2009)
5. Dutton, R.D.: Weak-heap sort. BIT 33(3), 372–381 (1993)
6. Floyd, R.W.: Algorithm 245: Treesort 3. Commun. ACM 7(12), 701 (1964)
7. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandra, S.: Cache-oblivious algorithms. In: 40th Annual Symposium on Foundations of Computer Science, pp. 285–297. IEEE Computer Society, Los Alamitos (1999)
8. Gonnet, G.H., Munro, J.I.: Heaps on heaps. SIAM J. Comput. 15(4), 964–971 (1986)
9. Katajainen, J., Vitale, F.: Navigation piles with applications to sorting, priority queues, and priority deques. Nordic J. Comput. 10(3), 238–262 (2003)
10. Kronrod, M.A.: Optimal ordering algorithm without operational field. Soviet Math. Dokl. 10, 744–746 (1969)
11. Li, Z., Reed, B.A.: Heap Building Bounds. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 14–23. Springer, Heidelberg (2005)
12. McDiarmid, C.J.H., Reed, B.A.: Building heaps fast. J. Algorithms 10(3), 352–365 (1989)
13. Vuillemin, J.: A data structure for manipulating priority queues. Commun. ACM 21(4), 309–315 (1978)
14. Wegener, I.: Bottom-up-heapsort, a new variant of heapsort beating, on an average, quicksort (if n is not very small). Theoret. Comput. Sci. 118(1), 81–98 (1993)
15. Wegener, I.: The worst case complexity of McDiarmid and Reed's variant of bottom-up heapsort is less than $n \log n + 1.1n$. Inform. and Comput. 97(1), 86–96 (1992)
16. Williams, J.W.J.: Algorithm 232: Heapsort. Commun. ACM 7(6), 347–348 (1964)

Model Checking Stochastic Branching Processes*

Taolue Chen, Klaus Dräger, and Stefan Kiefer

University of Oxford, UK
{taolue.chen,klaus.draeger,stefan.kiefer}@cs.ox.ac.uk

Abstract. Stochastic branching processes are a classical model for describing random trees, which have applications in numerous fields including biology, physics, and natural language processing. In particular, they have recently been proposed to describe parallel programs with stochastic process creation. In this paper, we consider the problem of model checking stochastic branching process. Given a branching process and a deterministic parity tree automaton, we are interested in computing the probability that the generated random tree is accepted by the automaton. We show that this probability can be compared with any rational number in PSPACE, and with 0 and 1 in polynomial time. In a second part, we suggest a tree extension of the logic PCTL, and develop a PSPACE algorithm for model checking a branching process against a formula of this logic. We also show that the qualitative fragment of this logic can be model checked in polynomial time.

1 Introduction

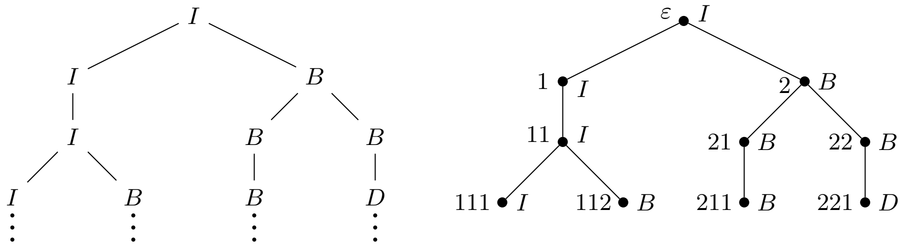
Consider an interactive program featuring two types of threads: interruptible threads (type I) and blocking threads (type B) which perform a non-interruptible computation or database transaction. An I-thread responds to user commands which occasionally trigger the creation of a B-thread. A B-thread may either terminate, or continue, or spawn another B-thread in an effort to perform its tasks in parallel. Under probabilistic assumptions on the thread behaviour, this scenario can be modelled as a *stochastic branching process* as follows:

$$\begin{array}{lll} I \overset{0.9}{\hookrightarrow} I & B \overset{0.2}{\hookrightarrow} D & D \overset{1}{\hookrightarrow} D \\ I \overset{0.1}{\hookrightarrow} (I,B) & B \overset{0.5}{\hookrightarrow} B & \\ & B \overset{0.3}{\hookrightarrow} (B,B) & \end{array} \tag{1}$$

This means, e.g., that a single step of an I-thread spawns a B-thread with probability 0.1. We have modelled the termination of a B-thread as a transformation

* This work was partially supported by the ERC Advanced Grant VERIWARE and EPSRC grant EP/F001096/1. Stefan Kiefer is supported by a DAAD postdoctoral fellowship.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 271–282, 2012.



(a) A prefix of a tree that the example process might create.

(b) A finite tree over $\{I, B, D\}$.

Fig. 1. Figures for Section 1 (left) and 2 (right)

into a "dead" state D.[1] A "run" of this process unravels an infinite tree whose branches record the computation of a thread and its ancestors. For example, Figure 1(a) shows the prefix of a tree that the example process might create. The probability of creating this tree prefix is the product of the probabilities of the applied rules, i.e., $0.1 \cdot 0.9 \cdot 0.1 \cdot 0.3 \cdot 0.5 \cdot 0.2$.

This example is an instance of a (stochastic multitype) branching process, which is a classical mathematical model with applications in numerous fields including biology, physics and natural language processing, see e.g. [12,2]. In [13] an extension of branching processes was introduced to model parallel programs with stochastic process creation. The broad applicability of branching processes arises from their simplicity: each *type* models a class of threads (or tasks, animals, infections, molecules, grammatical structures) with the same probabilistic behaviour.

This paper is about model checking the random trees created by branching processes. Consider a specification that requires a linear-time property to hold along all tree branches. In the example above, e.g., we may specify that "no process should become forever blocking", more formally, "on all branches of the tree we see infinitely many I or D". We would like to compute the probability that all branches satisfy such a given ω-regular word property. Curiously, this problem generalises two seemingly very different classical problems:

(i) If all rules in the branching process are of the form $X \xhookrightarrow{p} Y$, i.e., each node has exactly one child, the branching process describes a finite-state Markov chain. Computing the probability that a run of such a Markov chain satisfies an ω-regular property is a standard problem in probabilistic verification, see e.g. [5,16].

(ii) If for each type X in the branching process there is only one rule $X \xhookrightarrow{1} \alpha$ (where α is a nonempty sequence of types), then the branching process describes a unique infinite tree. Viewing the types in α as possible successor

[1] We disallow "terminating" rules like $B \xhookrightarrow{0.2} \varepsilon$. This is in contrast to classical branching processes, but technically more convenient for model checking, where absence of deadlocked states is customarily assumed.

states of X in a finite nondeterministic transition system, the branches in the created tree are exactly the possible runs in the finite transition systems. Of course, checking if all runs in such a transition system satisfy an ω-regular specification is also a well-understood problem.

One could expect that well-known Markov-chain based techniques for dealing with problem (i) can be generalised to branching processes. This is *not* the case: it follows from our results that in the example above, the probability that all branches satisfy the mentioned property is 0;[2] however, if the numbers 0.2 and 0.3 in (1) are swapped, the probability changes from 0 to 1. This is in sharp contrast to finite-state Markov chains, where qualitative properties (satisfaction with probability 0 resp. 1) do not depend on the exact probability of individual transitions.

The rules of a branching process are reminiscent of the rules of probabilistic pushdown automata (pPDA) or the equivalent model of recursive Markov chains (RMCs). However, the model-checking algorithms for both linear-time and branching-time logics proposed for RMCs and pPDAs [8,10,11] do *not* work for branching processes, essentially because pPDA and RMCs specify Markov chains, whereas branching processes specify random trees. Branching processes cannot be transformed to pPDAs, at least not in a straightforward way. Note that if the rules in the example above are understood as pPDA rules with I as starting symbol, then B will never even occur as the topmost symbol.

To model check branching processes, we must leave the realm of Markov chains and consider the probability space in terms of tree prefixes [12,2]. Consequently, we develop algorithms that are very different from the ones dealing with the special cases (i) and (ii) above. Nevertheless, for qualitative problems (satisfaction with probability 0 resp. 1) our algorithms also run in polynomial time with respect to the input models, even for branching processes that do not conform to the special cases (i) and (ii) above.

Instead of requiring a linear-time property to hold on all branches, we consider more general specifications in terms of *deterministic parity tree automata.* In a nutshell, our model-checking algorithms work as follows: (1) compute the "product" of the input branching process and the tree automaton; (2) reduce the analysis of the resulting product process to the problem of computing the probability that all branches reach a "good" symbol; (3) compute the latter probability by setting up and solving a nonlinear equation system. Step (1) can be seen as an instance of the automata-theoretic model-checking approach. The equation systems of step (3) are of the form $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{x}$ is a vector of variables, and $\boldsymbol{f}(\boldsymbol{x})$ is a vector of polynomials with nonnegative coefficients. Solutions to such equation systems can be computed or approximated efficiently [10,7,9]. Step (2) is, from a technical point of view, the main contribution of the paper; it requires a delicate and nontrivial analysis of the behaviour of branching processes.

In Section 4 we also consider logic specifications. We propose a new logic, PTTL, which relates to branching processes in the same manner as the logic PCTL relates to Markov chains. Recall that PCTL contains formulae such as

[2] Intuitively, this is because a B-thread more often clones itself than dies.

$[\phi \mathsf{U} \psi]_{\geq p}$ which specifies that the probability of runs satisfying $\phi \mathsf{U} \psi$ is at least p. For PTTL we replace the linear-time subformulae such as $\phi \mathsf{U} \psi$ with tree subformulae such as $\phi \mathsf{EU} \psi$ or $\phi \mathsf{AU} \psi$, so that, e.g., $[\phi \mathsf{EU} \psi]_{\geq p}$ specifies that the probability of trees that have a branch satisfying $\phi \mathsf{U} \psi$ is at least p, and $[\phi \mathsf{AU} \psi]_{\geq p}$ specifies that the probability of trees all whose branches satisfy $\phi \mathsf{U} \psi$ is at least p. We show that branching processes can be model checked against this logic in PSPACE, and against its qualitative fragment in polynomial time.

Related Work. The rich literature on branching processes (see e.g. [12,2] and the references therein) does not consider model-checking problems. Probabilistic split-join systems [13] are branching processes with additional features for process synchronisation and communication. The paper [13] focuses on performance measures (such as runtime, space and work), and does not provide a functional analysis. The models of pPDAs and RMCs also feature dynamic task creation by means of procedure calls, however, as discussed above, the existing model-checking algorithms [8,10,11] do not work for branching processes. Several recent works [10,7,9] have studied the exact and approximative solution of fixed-point equations of the above mentioned form. Our work connects these algorithms with the model-checking problem for branching processes.

Organisation of the Paper. After some preliminaries (Section 2), we present our results on parity specifications in Section 3. In Section 4 we propose the new logic PTTL and develop model-checking algorithms for it. We conclude in Section 5. Some proofs have been moved to a technical report [4].

2 Preliminaries

We let $\mathbb{N}$ and $\mathbb{N}_0$ denote the set of positive and nonnegative integers, respectively. Given a finite set Γ, we write $\Gamma^* := \bigcup_{k \in \mathbb{N}_0} \Gamma^k$ for the set of tuples and $\Gamma^+ := \bigcup_{k \in \mathbb{N}} \Gamma^k$ for the set of nonempty tuples over Γ.

Definition 1 (Branching process). *A* branching process *is a tuple $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ where Γ is a finite set of types, $\hookrightarrow \subseteq \Gamma \times \Gamma^+$ is a finite set of transition rules, Prob is a function assigning positive probabilities to transition rules so that for every $X \in \Gamma$ we have that $\sum_{X \hookrightarrow \alpha} \mathit{Prob}(X \hookrightarrow \alpha) = 1$.*

We write $X \overset{p}{\hookrightarrow} \alpha$ if $\mathit{Prob}(X \hookrightarrow \alpha) = p$. Observe that since the set of transition rules is finite, there is a global upper bound K_Δ such that $|\alpha| \leq K_\Delta$ for all $X \hookrightarrow \alpha$.

A *tree* is a nonempty prefix-closed language $V \subseteq \mathbb{N}^*$ for which there exists a function $\beta_V : V \to \mathbb{N}_0$ such that for all $w \in V$ and $k \in \mathbb{N}$, $wk \in V$ if and only if $k \leq \beta_V(w)$. $\beta_V(w)$ is called the *branching degree* of w in V. We denote by B_f the set of finite trees, and by B_i the set of infinite trees without leaves (i.e. trees such that $\beta_V(w) > 0$ for all $w \in V$). A *prefix* of V is a tree $V' \subseteq V$ such that for all $w \in V'$, $\beta_{V'}(w) \in \{0, \beta_V(w)\}$.

A *tree over* Γ is a pair (V, ℓ) where V is a tree, and $\ell : V \to \Gamma$ is a labelling function on the nodes. Given a tree $t = (V, \ell)$ with a node $u \in V$, we write $t_u = (V_u, \ell_u)$ for the subtree of t rooted at u; here $V_u = \{w \in \mathbb{N}^* \mid uw \in V\}$ and $\ell_u(w) = \ell(uw)$ for $w \in V_u$. A tree (V', ℓ') is a *prefix* of (V, ℓ) if V' is a prefix of V and $\ell'(w) = \ell(w)$ for all $w \in V'$.

A *path* (resp. *branch*) in a tree $t = (V, \ell)$ is a finite (resp. infinite) sequence $u_0, u_1, \ldots$ with $u_i \in V$ such that $u_0 = \epsilon$ is the root of t, and $u_{i+1} = u_i k_i$ for $k_i \in \mathbb{N}$ is a child of u_i. A *branch label* of t is a sequence $\ell(u_0), \ell(u_1), \ldots$, where $u_0, u_1, \ldots$ is a branch. The *successor word* of a node $w \in V$ is $\sigma_t(w) = \ell(w1) \ldots \ell(w\beta_V(w))$.

Given a tree $t = (V, \ell)$ over Γ and a subset $W \subseteq V$, we write $t \models \mathsf{AF}W$ if all its branches go through W, i.e., for all $v \in V$ there is a $w \in W$ such that v is a predecessor of w or vice versa. If $\Lambda \subseteq \Gamma$, we write $t \models \mathsf{AF}\Lambda$ for $t \models \mathsf{AF}\{w \in V \mid \ell(w) \in \Lambda\}$. Similarly, we write $t \models \mathsf{AG}\Lambda$ if $\ell(w) \in \Lambda$ for all $w \in V$.

Example 2. We illustrate these notions. Figure 1(b) shows a finite tree $t = (V, \ell) \in B_f$ over Γ with $\Gamma = \{I, B, D\}$ and $V = \{\varepsilon, 1, 11, 111, 112, 2, 21, 211, 22, 221\}$ and, e.g., $\ell(\varepsilon) = I$ and $\ell(112) = B$. We have $\beta_V(\varepsilon) = 2$ and $\beta_V(21) = 1$ and $\beta_V(211) = 0$. The node 2 is a predecessor of 211. The tree $t' = (V', \ell')$ with $V' = \{\varepsilon, 1, 2, 21, 22\}$ and ℓ' being the restriction of ℓ on V' is a prefix of t. The sequence $\varepsilon, 2, 21$ is a path in t. We have $\sigma_t(11) = IB$. The tree satisfies $t \models \mathsf{AF}\{1, 21, 221\}$ and $t \models \mathsf{AF}\{I\}$.

A tree $t = (V, \ell)$ over Γ is *generated* by a branching process $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ if for every $w \in V$ with $\beta_V(w) > 0$ we have $\ell(w) \hookrightarrow \sigma_t(w)$. We write $(\!|\Delta|\!)$ and $[\![\Delta]\!]$ for the sets of trees (V, ℓ) generated by Δ with $V \in B_f$ and $V \in B_i$, respectively. For any $X \in \Gamma$, $(\!|\Delta|\!)_X \subseteq (\!|\Delta|\!)$ and $[\![\Delta]\!]_X \subseteq [\![\Delta]\!]$ contain those trees (V, ℓ) for which $\ell(\epsilon) = X$.

Definition 3 (Probability space of trees, cf. [12, Chap. VI]). *Let $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ be a branching process. For any finite tree $t = (V, \ell) \in (\!|\Delta|\!)$, let the* cylinder *over t be $\mathit{Cyl}_\Delta(t) := \{t' \in [\![\Delta]\!] \mid t$ is a prefix of $t'\}$, and define $p_\Delta(t) := \prod_{w \in V : \beta_V(w) > 0} \mathit{Prob}(\ell(w), \sigma_t(w))$. For each $X \in \Gamma$ we define a probability space $([\![\Delta]\!]_X, \Sigma_X, \Pr_X)$, where Σ_X is the σ-algebra generated by $\{\mathit{Cyl}_\Delta(t) \mid t \in (\!|\Delta|\!)_X\}$, and $\Pr_X$ is the probability measure generated by $\Pr_X(\mathit{Cyl}_\Delta(t)) = p_\Delta(t)$. Sometimes we write $\Pr_X^\Delta$ to indicate Δ. We may drop the subscript of $\Pr_X$ if X is understood. We often write t_X to mean a tree $t \in [\![\Delta]\!]_X$ randomly sampled according to the probability space above.*

Example 4. Let $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ be the branching process with $\Gamma = \{I, B, D\}$ and the rules as given in (1) on page 271. The tree t from Figure 1(b) is generated by Δ: we have $t \in (\!|\Delta|\!)_I$. We have $\Pr_I(\mathit{Cyl}_\Delta(t)) = p_\Delta(t) = 0.1 \cdot 0.9 \cdot 0.1 \cdot 0.3 \cdot 0.5 \cdot 0.2$; this is probability of those trees $t' \in [\![\Delta]\!]_I$ that have prefix t.

We say that a quantity $q \in [0, 1]$ is *PPS-expressible* if one can compute, in polynomial time, an integer $m \in \mathbb{N}$ and a fixed-point equation system $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{x}$ is a vector of m variables, $\boldsymbol{f}$ is a vector of m multivariate polynomials over $\boldsymbol{x}$ with nonnegative rational coefficients, $\boldsymbol{f}(\mathbf{1}) \leq \mathbf{1}$ where $\mathbf{1}$ denotes the vector $(1, \ldots, 1)$, and q is the first component of the least nonnegative solution $\boldsymbol{y} \in [0, \infty)^m$ of $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$.

Proposition 5. *Let q be PPS-expressible. We have:*

(a) For $\tau \in \{0, 1\}$ one can decide in (strongly) polynomial time whether $q = \tau$.
(b) For $\tau \in \mathbb{Q}$ one can decide in polynomial space whether $q \bowtie \tau$, where $\bowtie \in \{<, >, \leq, \geq, =, \neq\}$.
(c) One can approximate q within additive error 2^{-j} in time polynomial in both j and the (binary) representation size of $\boldsymbol{f}$.

Part (a) follows from [10,6]. Part (b) is shown in [10, section 4] by appealing to the existential fragment of the first-order theory of the reals, which is decidable in PSPACE, see [3,14]. Part (c) follows from a recent result [9, Corollary 4.5]. The following proposition follows from a classical result on branching processes [12].

Proposition 6. *Let $\Delta = (\Gamma, \hookrightarrow, Prob)$ be a branching process. Let $X \in \Gamma$ and $\Lambda \subseteq \Gamma$. Then $\Pr[t_X \models \mathsf{AF}\Lambda]$ is PPS-expressible.*

3 Parity Specifications

In this section we show how to compute the probability of those trees that satisfy a given parity specification.

A *(top-down) deterministic (amorphous) parity tree automaton* (DPTA) is a tuple $\mathcal{A} = (Q, \Gamma, q_0, \delta, c)$, where Q is the finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Gamma \times \mathbb{N} \to Q^*$ is the transition function satisfying $|\delta(q, X, n)| = n$ for all q, X, n, and $c : Q \to \mathbb{N}$ is a colouring function. Such an automaton $\mathcal{A}$ maps a tree $t = (V, \ell)$ over Γ to the (unique) tree $\mathcal{A}(t) = (V, \ell')$ over Q such that $\ell(\varepsilon) = q_0$ and for all $w \in V$, $\sigma_{\mathcal{A}(t)}(w) = \delta(\ell'(w), \ell(w), \beta_V(w))$.

Automaton $\mathcal{A} = (Q, \Gamma, q_0, \delta, c)$ *accepts* a tree t over Γ if for all branch labels $q_0 q_1 \cdots \in Q^\omega$ of $\mathcal{A}(t)$ the highest colour that occurs infinitely often in $c(q_0), c(q_1), \ldots$ is even.

Example 7. Recall (e.g., from [15]) that any ω-regular *word* property (e.g., any LTL specification) can be translated into a deterministic parity *word* automaton. Such an automaton, in turn, can be easily translated into a DPTA which specifies that the labels of *all* branches satisfy the ω-regular word property. We do not spell out the translation, but let us note that in the resulting tree automaton, for all $(q, X) \in Q \times \Gamma$ there is $q' \in Q$ such that $\delta(q, X, k) = (q', \ldots, q')$ for all k.

Given a colouring function $c : \Gamma \to \mathbb{N}$, a tree (V, ℓ) over Γ is called *good* for c if for each branch $u_0, u_1, \cdots$ the largest number that occurs infinitely often in the sequence $c(\ell(u_0)), c(\ell(u_1)), \ldots$ is even. The following proposition is immediate.

Proposition 8. *Let $\Delta = (\Gamma, \hookrightarrow, Prob)$ be a branching process, and let $\mathcal{A} = (Q, \Gamma, q_0, \delta, c)$ be a DPTA. Define the* product *of Δ and $\mathcal{A}$ as the branching process $\Delta_\bullet = (\Gamma \times Q, \hookrightarrow_\bullet, Prob_\bullet)$ with $(X, q) \overset{p}{\hookrightarrow}_\bullet (Y_1, q_1) \ldots (Y_k, q_k))$ for $X \overset{p}{\hookrightarrow}$*

$Y_1 \ldots Y_k$, where $(q_1, \ldots, q_k) = \delta(q, X, k)$. Define $c_\bullet : \Gamma \times Q \to \mathbb{N}$ by $c_\bullet(X, q) := c(q)$. Then for all $X \in \Gamma$ we have

$$\Pr^{\Delta}_X[t \textit{ is accepted by } \mathcal{A}] = \Pr^{\Delta_\bullet}_{(X,q_0)}[t \textit{ is good for } c_\bullet]\,.$$

In view of Proposition 8, it suffices to compute the probability $\Pr[t_X \text{ is good for } c]$, where a branching process $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ with $X \in \Gamma$ and a colouring function $c : \Gamma \to \mathbb{N}$ are fixed for the rest of the section. We write $\Pr[t_X \text{ is good}]$ if c is understood. We distinguish between the *qualitative problem*, i.e., computing whether $\Pr[t_X \text{ is good}] = 1$ holds for a given $X \in \Gamma$, and the *quantitative problem*, i.e., the computation of $\Pr[t_X \text{ is good}]$.

3.1 The Qualitative Problem

The outline of this subsection is the following: We will show that the qualitative problem can be solved in polynomial time (Theorem 12). First we show (Proposition 9) that it suffices to compute all *clean* types, where “clean” is defined below. We will show (Lemma 11) that a type X is clean if and only if $\Pr[t_X \models \mathsf{AF}\Lambda] = 1$ holds for suitable set $\Lambda \subseteq \Gamma$. By Proposition 6 the latter condition can be checked in polynomial time, completing the qualitative problem.

If there exists a tree $(V, \ell) \in [\![\Delta]\!]_X$ and a node $u \in V$ with $\ell(u) = Y$, then we say that Y is *reachable* from X. Given $X \in \Gamma$ and a finite word $w = X_0 \cdots X_m \in \Gamma^+$, we say that w is *X-closing* if $m \geq 1$ and $X_m = X$ and $c(X_i) \leq c(X)$ for $0 \leq i \leq m$. A branch with label $X_0 X_1 \cdots \in \Gamma^\omega$ is called *X-branch* if $X_0 = X$ and there is a sequence $0 = m_0 < m_1 < m_2 < \cdots$ such that $X_{m_i} \cdots X_{m_{i+1}}$ is X-closing for all $i \in \mathbb{N}$. We say that a type $Y \in \Gamma$ is *odd* (resp. *even*), if $c(Y)$ is odd (resp. even). Observe that a tree t is good if and only if for all its vertices u and all *odd* types Y the subtree t_u does not have a Y-branch. A type $Y \in \Gamma$ is *clean* if Y is even or $\Pr[t_Y \text{ has a } Y\text{-branch}] = 0$. The following proposition reduces the qualitative problem to the computation of all clean types.

Proposition 9. *We have that $\Pr[t_X$ is good$] = 1$ if and only if all Y reachable from X are clean.*

Proof. If there is an unclean reachable Y, then $\Pr[t_Y \text{ is good}] < 1$ and so $\Pr[t_X \text{ is good}] < 1$. Otherwise, for each node v in t_X and for each odd Y we have that $\Pr[(t_X)_v \text{ has a } Y\text{-branch}] = 0$. Since the set of nodes in a tree is countable, it follows that almost surely no subtree of t_X has a Y-branch for odd Y; i.e., t_X is almost surely good. □

Call a path in a tree *X-closing* if the corresponding label sequence is X-closing. Given $X \in \Gamma$, we define

$$N_X := \{Y \in \Gamma \mid \text{ no tree in } [\![\Delta]\!]_Y \text{ has an } X\text{-closing path}\}\,.$$

Note that $c(Y) > c(X)$ implies $Y \in N_X$ and that N_X is computable in polynomial time. A word $X_0 X_1 \cdots \in (\Gamma^* \cup \Gamma^\omega)$ is called *X-failing* if no prefix is

X-closing and there is $i \geq 0$ with $X_i \in N_X$. A branch in a tree is called X-failing if the corresponding branch label is X-failing. Given $X \in \mathit{\Gamma}$ and a tree t, let $\mathsf{Clos}_X(t)$ (resp. $\mathsf{Fail}_X(t)$) denote the set of those nodes w in t such that the path to w is X-closing (resp. X-failing) and no proper prefix of this path is X-closing (resp. X-failing). We will need the following lemma.

Lemma 10. *Define the events* $C := \{t_X \mid t_X \models \mathsf{AF}\,(\mathsf{Clos}_X(t_X) \cup \mathsf{Fail}_X(t_X))\}$ *and* $I := \{t_X \mid \mathsf{Clos}_X(t_X)$ *is infinite*$\}$. *Then* $C \cap I = \emptyset$ *and* $\Pr[C \cup I] = 1$.

The following lemma states in particular that an odd type X is clean if and only if $\Pr[t_X \models \mathsf{AF}N_X] = 1$. We prove something slightly stronger:

Lemma 11. *Define the events* $F := \{t_X \mid t_X \models \mathsf{AF}N_X\}$ *and* $H := \{t_X \mid t_X$ *has an* X*-branch*$\}$. *Then* $F \cap H = \emptyset$ *and* $\Pr[F \cup H] = 1$.

Now we have:

Theorem 12. *One can decide in polynomial time whether* $\Pr[t_X$ *is good*$] = 1$.

Proof. By Proposition 9 it suffices to show that cleanness can be determined in polynomial time. By Lemma 11 an odd type X is clean if and only if $\Pr[t_X \models \mathsf{AF}N_X] = 1$. The latter condition is decidable in polynomial time by Proposition 6. □

Example 13. Consider the branching process with $\mathit{\Gamma} = \{1, 2, 3, 4\}$ and the rules $1 \xhookrightarrow{1/3} 11$, $1 \xhookrightarrow{2/3} 4$, $2 \xhookrightarrow{1/2} 13$, $2 \xhookrightarrow{1/2} 23$, $3 \xhookrightarrow{2/3} 33$, $3 \xhookrightarrow{1/3} 1$, $4 \xhookrightarrow{1} 4$, and the colouring function c with $c(i) = i$ for $i \in \{1, 2, 3, 4\}$. Using a simple reachability analysis one can compute the sets $N_1 = \{2, 3, 4\}$, $N_2 = \{1, 3, 4\}$, $N_3 = \{1, 4\}$, $N_4 = \emptyset$. Applying Proposition 6 we find $\Pr[t_3 \models \mathsf{AF}N_3] < 1 = \Pr[t_1 \models \mathsf{AF}N_1]$. It follows by Lemma 11 that the only *un*clean type is 3. Since type 3 is only reachable from 2 and from 3, Proposition 9 implies that $\Pr[t_X$ is good$] = 1$ holds if and only if $X \in \{1, 4\}$.

3.2 The Quantitative Problem

Define $G := \{X \in \mathit{\Gamma} \mid$ all Y reachable from X are clean$\}$. The following Proposition 14 states that $\Pr[t_X$ is good$] = \Pr[t_X \models \mathsf{AF}G]$. This implies, by Proposition 6, that the probability is PPS-expressible (see Theorem 15).

Proposition 14. *We have* $\Pr[t_X$ *is good*$] = \Pr[t_X \models \mathsf{AF}G]$.

This implies the following theorem.

Theorem 15. *For any* $X \in \mathit{\Gamma}$ *we have that* $\Pr[t_X$ *is good*$]$ *is PPS-expressible.*

Proof. By Proposition 14 we have $\Pr[t_X$ is good$] = \Pr[t_X \models \mathsf{AF}G]$. So we can apply Proposition 6 with $\mathit{\Lambda} := G$. Note that G can be computed in polynomial time, as argued in the proof of Theorem 12. □

Example 16. We continue Example 13, where we have effectively computed $G = \{1, 4\}$, and thus established that $\Pr[t_1 \text{ is good}] = \Pr[t_4 \text{ is good}] = 1$. By Proposition 14 the probabilities $\Pr[t_2 \text{ is good}]$ and $\Pr[t_3 \text{ is good}]$ are given by $\Pr[t_2 \models \mathsf{AF}G]$ and $\Pr[t_3 \models \mathsf{AF}G]$. Proposition 6 assures that these probabilities are PPS-expressible; in fact they are given by the least nonnegative solution of the equation system $[x_2 = \frac{1}{2}x_3 + \frac{1}{2}x_2x_3,\ x_3 = \frac{2}{3}x_3^2 + \frac{1}{3}]$, which is $x_2 = \frac{1}{3}$ and $x_3 = \frac{1}{2}$. Hence, we have $\Pr[t_2 \text{ is good}] = \frac{1}{3}$ and $\Pr[t_3 \text{ is good}] = \frac{1}{2}$.

A Lower Bound. We close the section with a hardness result in terms of the PosSLP problem, which asks whether a given straight-line program or, equivalently, arithmetic circuit with operations $+$, $-$, $\cdot$, and inputs 0 and 1, and a designated output gate, outputs a positive integer or not. PosSLP is in PSPACE, but known to be in NP. The PosSLP problem is a fundamental problem for numerical computation, see [1] for more details.

For given Γ with $D \in \Gamma$, consider the DPTA $\mathcal{A}_{hit} = (\{q, r\}, \Gamma, a, \delta, c)$ with $c(q) = 1$ and $c(r) = 2$; $\delta(q, X, 1) = (q)$ and $\delta(q, X, 2) = (q, q)$ for $X \in \Gamma \setminus \{D\}$; $\delta(q, D, 1) = (r)$ and $\delta(q, D, 2) = (r, r)$; $\delta(r, X, 1) = (r)$ and $\delta(r, X, 2) = (r, r)$ for $X \in \Gamma$. Automaton $\mathcal{A}_{hit}$ specifies that all branches satisfy the LTL property $\mathsf{F}D$, i.e., all branches eventually hit D. Let QUANT-HIT denote the problem to decide whether $\Pr^{\Delta}_{X}[t \text{ is accepted by } \mathcal{A}_{hit}] > p$ holds for a given branching process $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ with $X \in \Gamma$ and a given rational $p \in (0, 1)$. By Theorem 15 and Proposition 5, QUANT-HIT is in PSPACE. We have the following proposition:

Proposition 17 (see Theorem 5.3 of [10]). *QUANT-HIT is PosSLP-hard.*

4 Logic Specifications

In this section, we propose a logic akin to PCTL, called *probabilistic tree temporal logic*, to specify the properties of random trees generated from a branching process. We also present model-checking algorithms for this logic.

Definition 18 (PTTL). Probabilistic Tree Temporal Logic (PTTL) *formulae over a set Σ of atomic propositions are defined by the following grammar:*

$$\begin{aligned} \phi, \phi' &::= \top \mid a \mid \neg\phi \mid \phi \wedge \phi' \mid [\psi]_{\bowtie r} \\ \psi &::= \mathsf{AX}\phi \mid \mathsf{EX}\phi \mid \phi\mathsf{AU}\phi' \mid \phi\mathsf{EU}\phi' \mid \phi\mathsf{AR}\phi' \mid \phi\mathsf{ER}\phi' \,, \end{aligned}$$

where $a \in \Sigma$, $\bowtie \in \{<, \leq, \geq, >\}$, and $r \in \mathbb{Q} \cap [0, 1]$. If $r \in \{0, 1\}$ holds for all subformulae of a PTTL formula ϕ, we say that ϕ is in the qualitative fragment *of PTTL. We use standard abbreviations such as $\bot$ for $\neg\top$, $\mathsf{AF}\phi$ for $\top\mathsf{AU}\phi$, $\mathsf{EG}\phi$ for $\bot\mathsf{ER}\phi$, etc.*

For the PTTL semantics we need the notion of a *labelled* branching process, which is a branching process $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ extended by a function $\chi : \Gamma \to 2^{\Sigma}$, where $\chi(X)$ indicates which atomic propositions the type X satisfies.

Definition 19 (Semantics of PTTL). *Given a labelled branching process $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob}, \chi)$, we inductively define a* satisfaction relation $\models$ *as follows, where $X \in \Gamma$:*

$$\begin{array}{ll}
X \models \top & \\
X \models a & \Leftrightarrow a \in \chi(X) \\
X \models \neg\phi & \Leftrightarrow X \not\models \phi \\
X \models \phi \wedge \phi' & \Leftrightarrow X \models \phi \textit{ and } X \models \phi' \\
X \models [\psi]_{\bowtie r} & \Leftrightarrow \Pr_X^{\Delta}[t_X \models \psi] \bowtie r \\
t \models \mathsf{AX}\phi & \Leftrightarrow \textit{for all branches } u_0 u_1 \cdots \textit{ of } t \textit{ we have } \ell(u_1) \models \phi \\
t \models \phi\mathsf{AU}\phi' & \Leftrightarrow \textit{for all branches } u_0 u_1 \cdots \textit{ of } t \textit{ there exists } i \in \mathbb{N} \textit{ with} \\
 & \quad\ \ell(u_i) \models \phi' \textit{ and for all } 0 \le j < i \textit{ we have } \ell(u_j) \models \phi \\
t \models \phi\mathsf{AR}\phi' & \Leftrightarrow \textit{for all branches } u_0 u_1 \cdots \textit{ of } t \textit{ and for all } i \in \mathbb{N} \textit{ we have} \\
 & \quad\ \ell(u_i) \models \phi' \textit{ or there exists } 0 \le j < i \textit{ with } \ell(u_j) \models \phi
\end{array}$$

The modalities EX*,* EU *and* ER *are defined similarly, with "for all branches" replaced by "there exists a branch".*

We now present the model checking algorithm. The algorithm shares its basic structure with the well-known algorithm for (P)CTL and finite (probabilistic) transition systems. Given a PTTL formula ϕ, the algorithm recursively evaluates the truth values of the PTTL subformulae ψ of ϕ for all types. The boolean operators can be dealt with as in the CTL algorithm. Hence, it suffices to examine formulae of the form $[\psi]_{\bowtie r}$. Observe that we have $\mathsf{EX}\phi \equiv \neg\mathsf{AX}\neg\phi$ and $\phi\mathsf{ER}\phi' \equiv \neg(\neg\phi\mathsf{AU}\neg\phi')$ and $\phi\mathsf{EU}\phi' \equiv \neg(\neg\phi\mathsf{AR}\neg\phi')$ and

$$X \models [\neg\phi]_{\bowtie r} \text{ if and only if } X \models [\phi]_{\overline{\bowtie} 1-r}\,,$$

where $\overline{\bowtie} \in \{\geq, >, <, \leq\}$ is the complement operator of $\bowtie \in \{<, \leq, \geq, >\}$. Hence, it suffices to deal with the following three cases: (i) $X \models [\mathsf{AX}\phi]_{\bowtie r}$; (ii) $X \models [\phi\mathsf{AU}\psi]_{\bowtie r}$; (iii) $X \models [\phi\mathsf{AR}\psi]_{\bowtie r}$. We assume in the following case distinction that the algorithm has already computed the truth values of the subformulae ϕ, ψ. One could now construct a suitable DPTA for each of the cases (i)–(iii), and proceed according to the machinery of Section 3. Instead we present in the following a more direct and more efficient algorithm which takes advantage of the special shape of the linear-time operators X, U and R.

Case (i): We have $\Pr[t_X \models \mathsf{AX}\phi] = \sum_{\substack{X \overset{p}{\hookrightarrow} Y_1 \ldots Y_k \\ Y_1, \ldots, Y_k \models \phi}} p$, which is easy to compute. So one can decide in polynomial time whether $X \models [\mathsf{AX}\phi]_{\bowtie r}$.

Case (ii): We reduce the check of the $\phi\mathsf{AU}\psi$ modality to a check of AF. To this end, we define a branching process $\Delta' = (\Gamma \times \{0, \frac{1}{2}, 1\}, \hookrightarrow', \mathit{Prob}')$ which tracks the "status" of $\phi\mathsf{AU}\psi$. We define Δ' in terms of an auxiliary function $f_{\phi,\psi} : \Gamma \to \{0, \frac{1}{2}, 1\}$ with $f_{\phi,\psi}(Y) = 0$ if $Y \models \neg\phi \wedge \neg\psi$, $f_{\phi,\psi}(Y) = \frac{1}{2}$ if $Y \models \phi \wedge \neg\psi$, and $f_{\phi,\psi}(Y) = 1$ if $Y \models \psi$. For any rule $X \overset{p}{\hookrightarrow} Y_1 \ldots Y_k$ in Δ, there are three corresponding rules in Δ', namely $(X, 0) \overset{p}{\hookrightarrow} (Y_1, 0) \ldots (Y_k, 0)$,

$(X,1) \overset{p}{\hookrightarrow} (Y_1,1)\ldots(Y_k,1)$, and $(X,\frac{1}{2}) \overset{p}{\hookrightarrow} (Y_1, f_{\phi,\psi}(Y_1))\ldots(Y_k, f_{\phi,\psi}(Y_k))$. By this construction we achieve $\Pr_X^{\Delta}[t_X \models \phi \mathsf{AU} \psi] = \Pr_{X'}^{\Delta'}[t_{X'} \models \mathsf{AF}\Lambda]$ for $X' = (X, f_{\phi,\psi}(X))$ and $\Lambda := \Gamma \times \{1\}$. Hence, using Propositions 5 and 6 we obtain that whether $X \models [\phi \mathsf{AU} \psi]_{\bowtie r}$ holds is decidable in PSPACE; and in polynomial time for $r \in \{0,1\}$.

Case (iii): Similarly to case (ii) we reduce the check of $\phi \mathsf{AR} \psi$ to a check of AG. This time we define $\Delta' = (\Gamma \times \{0,\frac{1}{2},1\}, \hookrightarrow', \mathit{Prob}')$ in terms of an auxiliary function $g_{\phi,\psi} : \Gamma \to \{0,\frac{1}{2},1\}$ with $g_{\phi,\psi}(Y) = 0$ if $Y \models \neg\psi$, $g_{\phi,\psi}(Y) = \frac{1}{2}$ if $Y \models \neg\phi \;\wedge\; \psi$, $g_{\phi,\psi}(Y) = 1$ if $Y \models \phi \;\wedge\; \psi$. The rules of Δ' are defined as in case (ii), except that $f_{\phi,\psi}$ is replaced with $g_{\phi,\psi}$. By this construction we achieve $\Pr_X^{\Delta}[t_X \models \phi \mathsf{AR} \psi] = \Pr_{X'}^{\Delta'}[t_{X'} \models \mathsf{AG}\Lambda]$ for $X' = (X, g_{\phi,\psi}(X))$ and $\Lambda := \Gamma \times \{\frac{1}{2}, 1\}$. The following lemma allows to express this probability in terms of AF instead of AG:

Lemma 20. *Let $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ be a branching process. Let $\Lambda \subseteq \Gamma$ such that no type in Λ is reachable from any type in $\Gamma \setminus \Lambda$. Define $G := \{Y \in \Lambda \mid$ all types reachable from Y are in $\Lambda\}$. Let $X \in \Gamma$. Then $\Pr[t_X \models \mathsf{AG}\Lambda] = \Pr[t_X \models \mathsf{AF}G]$.*

To summarize case (iii): we have reduced AR to AG and then AG to AF. Hence, using Propositions 5 and 6 we obtain that whether $X \models [\phi \mathsf{AR} \psi]_{\bowtie r}$ holds is decidable in PSPACE; and in polynomial time for $r \in \{0,1\}$.

As the overall algorithm computes the truth values of the subformulae recursively, we have proved the following theorem:

Theorem 21. *Model checking branching processes against PTTL is in* PSPACE*. Model checking branching processes against the qualitative fragment of PTTL is in* P*.*

5 Conclusions and Future Work

Branching processes are a basic formalism for modelling probabilistic parallel programs with dynamic process creation. This paper is the first to consider the verification of branching processes, We have shown how to model check specifications given in terms of deterministic parity automata, a problem that unifies and strictly generalises linear-time model-checking problems for Markov chains and for (nonprobabilistic) nondeterministic transition systems. We have also provided model-checking algorithms for a new logic, PTTL, suitable for specifying probabilistic properties of random trees. To obtain these results we have provided reductions to computing the probability of hitting "good" states along all branches.

Future research in this area should involve:

- the complexity of the problem where the specification is an LTL formula required to hold on all branches;
- the problem where deterministic parity automata are replaced by other tree specification formalisms, such as CTL (or CTL*) formulae;

- extending the model-checking algorithms to accommodate the synchronisation and communication features of probabilistic split-join systems.

It seems that at least the latter two problems require additional techniques, as the children of a node in the branching process can no longer be treated independently.

Acknowledgements. We thank anonymous reviewers for their valuable feedback.

References

1. Allender, E., Bürgisser, P., Kjeldgaard-Pedersen, J., Miltersen, P.B.: On the complexity of numerical analysis. In: IEEE Conference on Computational Complexity, pp. 331–339 (2006)
2. Athreya, K.B., Ney, P.E.: Branching Processes. Springer (1972)
3. Canny, J.: Some algebraic and geometric computations in PSPACE. In: STOC 1988, pp. 460–467 (1988)
4. Chen, T., Dräger, K., Kiefer, S.: Model checking stochastic branching processes. Technical report, arxiv.org (2012), http://arxiv.org/abs/1206.1317
5. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. Journal of the ACM 42, 857–907 (1995)
6. Esparza, J., Gaiser, A., Kiefer, S.: Computing least fixed points of probabilistic systems of polynomials. In: Proceedings of STACS, pp. 359–370 (2010)
7. Esparza, J., Kiefer, S., Luttenberger, M.: Computing the least fixed point of positive polynomial systems. SIAM Journal on Computing 39(6), 2282–2335 (2010)
8. Esparza, J., Kučera, A., Mayr, R.: Model checking probabilistic pushdown automata. In: LICS 2004, pp. 12–21. IEEE (2004)
9. Etessami, K., Stewart, A., Yannakakis, M.: Polynomial-time algorithms for multitype branching processes and stochastic context-free grammars. In: Proceedings of STOC, pp. 579–588 (2012)
10. Etessami, K., Yannakakis, M.: Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. Journal of the ACM 56(1), 1–66 (2009)
11. Etessami, K., Yannakakis, M.: Model checking of recursive probabilistic systems. ACM Transactions on Computational Logic 13(2) (to appear 2012)
12. Harris, T.E.: The Theory of Branching Processes. Springer (1963)
13. Kiefer, S., Wojtczak, D.: On Probabilistic Parallel Programs with Process Creation and Synchronisation. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 296–310. Springer, Heidelberg (2011)
14. Renegar, J.: On the computational complexity and geometry of the first-order theory of the reals. Parts I–III. Journal of Symbolic Computation 13(3), 255–352 (1992)
15. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Beyond Words, vol. 3, pp. 389–455. Springer (1997)
16. Vardi, M.Y.: Probabilistic Linear-Time Model Checking: An Overview of the Automata-Theoretic Approach. In: Katoen, J.-P. (ed.) AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999. LNCS, vol. 1601, pp. 265–276. Springer, Heidelberg (1999)

Parameterized Study of the Test Cover Problem

Robert Crowston[1], Gregory Gutin[1], Mark Jones[1],
Saket Saurabh[2], and Anders Yeo[3]

[1] Royal Holloway, University of London
Egham TW20 0EX, UK
{robert,gutin,markj}@cs.rhul.ac.uk
[2] The Institute of Mathematical Sciences
Chennai 600 113, India
saket@imsc.res.in
[3] University of Johannesburg
Auckland Park, 2006 South Africa
anders.yeo.work@gmail.com

Abstract. In this paper we carry out a systematic study of a natural covering problem, used for identification across several areas, in the realm of parameterized complexity. In the Test Cover problem we are given a set $[n] = \{1, \dots, n\}$ of items together with a collection, $\mathcal{T}$, of distinct subsets of these items called tests. We assume that $\mathcal{T}$ is a test cover, i.e., for each pair of items there is a test in $\mathcal{T}$ containing exactly one of these items. The objective is to find a minimum size subcollection of $\mathcal{T}$, which is still a test cover. The generic parameterized version of Test Cover is denoted by $p(k, n, |\mathcal{T}|)$-Test Cover. Here, we are given $([n], \mathcal{T})$ and a positive integer parameter k as input and the objective is to decide whether there is a test cover of size at most $p(k, n, |\mathcal{T}|)$. We study four parameterizations for Test Cover and obtain the following:

(a) k-Test Cover, and $(n-k)$-Test Cover are fixed-parameter tractable (FPT), i.e., these problems can be solved by algorithms of runtime $f(k) \cdot poly(n, |\mathcal{T}|)$, where $f(k)$ is a function of k only.
(b) $(|\mathcal{T}| - k)$-Test Cover and $(\log n + k)$-Test Cover are W[1]-hard. Thus, it is unlikely that these problems are FPT.

1 Introduction

The input to the Test Cover problem consists of a set of items, $[n] := \{1, 2, \dots, n\}$, and a collection of distinct sets, $\mathcal{T} = \{T_1, \dots, T_m\}$, called *tests*. We say that a test T_q *separates* a pair i, j of items if $|\{i, j\} \cap T_q| = 1$. Subcollection $\mathcal{T}' \subseteq \mathcal{T}$ is a *test cover* if each pair i, j of distinct items is separated by a test in $\mathcal{T}'$. The objective is to find a test cover of minimum size, if one exists. Since it is easy to decide, in polynomial time, whether the collection $\mathcal{T}$ itself is a test cover, henceforth we will assume that $\mathcal{T}$ is a test cover.

Test Cover arises naturally in the following general setting of identification problems: Given a set of items and a set of binary attributes that may or may not occur in each item, the aim is to find the minimum size subset of attributes

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 283–295, 2012.

(corresponding to a minimum test cover) such that each item can be uniquely identified from the information on which of this subset of attributes it contains. Test Cover arises in fault analysis, medical diagnostics, pattern recognition, and biological identification (see, e.g., [11,12,16]).

The Test Cover problem has been also studied extensively from an algorithmic view point. The problem is NP-hard, as was shown by Garey and Johnson [7]. Moreover, Test Cover is APX-hard [11]. There is an $O(\log n)$-approximation algorithm for the problem [16] and there is no $o(\log n)$-approximation algorithm unless P=NP [11]. These approximation results are obtained using reductions from Test Cover to the well-studied Set Cover problem, where given a collection $\mathcal{S}$ of subsets of $[n]$ *covering* $[n]$ (i.e., $\cup_{X\in\mathcal{S}}X=[n]$) and integer t, we are to decide whether there is a subcollection of $\mathcal{S}$ of size t covering $[n]$.

In this paper we carry out a systematic study of Test Cover in the realm of parameterized complexity[1]. The following will be a generic parameterization of the problem:

$p(k,n,m)$-Test Cover
Instance: A set $\mathcal{T}$ of m tests on $[n]$ such that $\mathcal{T}$ is a test cover.
Parameter: k.
Question: Does $\mathcal{T}$ have a test cover with at most $p(k,n,m)$ tests?

We will consider four parameterizations of Test Cover: k-Test Cover, $(m-k)$-Test Cover, $(n-k)$-Test Cover, and $(\log n+k)$-Test Cover. The first parameterization is standard; its complexity is not hard to establish since it is well-known that there is no test cover of size less than $\lceil\log n\rceil$ [11] and the bound is tight. This bound suggests the parameterization $(\log n+k)$-Test Cover (above a tight lower bound). The parameterization $(m-k)$-Test Cover is a natural parameterization below a tight upper bound. There is always a test cover of size at most $n-1$ [2] and $\mathcal{T}=\{\{1\},\dots,\{n-1\}\}$ shows that the bound is tight. Thus, $(n-k)$-Test Cover is another parameterization below a tight upper bound.

In this paper, we will use some special cases of the following generic parameterization of Set Cover:

$p(k,n,m)$-Set Cover
Instance: A collection $\mathcal{S}$ of m subsets of $[n]$ covering $[n]$.
Parameter: k.
Question: Does $\mathcal{S}$ contain a subcollection of size $p(k,n,m)$ covering $[n]$?

Three of our parameterizations for Test Cover are below or above guaranteed lower or upper bounds. The study of parameterized problems above a guaranteed lower/upper bound was initiated by Mahajan and Raman [14]. They showed that some above guarantee versions of Max Cut and Max Sat are FPT; in the case of Max Sat the input is a CNF formula with m clauses together with an integer

[1] Basic notions on parameterized complexity are given in the end of this section.

k (the parameter) and the question is whether there exists an assignment that satisfies at least $m/2 + k$ clauses. Later, Mahajan et al. [15] published a paper with several new results and open problems around parameterizations beyond guaranteed lower and upper bounds. In a breakthrough paper Gutin et al. [9] developed a probabilistic approach to problems parameterized above or below tight bounds. Alon et al. [1] combined this approach with a method from Fourier analysis to obtain an FPT algorithm for parameterized MAX r-SAT beyond the guaranteed lower bound. In the same paper a quadratic kernel was also given for MAX r-SAT. Other significant results in this direction include quadratic kernels for ternary permutation constraint satisfaction problems parameterized above average and results on systems of linear equations over the field of two elements [3,4,10].

We establish parameterized complexity of all four parameterizations of TEST COVER:

(i) Since there is no test cover of size less than $\lceil \log n \rceil$, k-TEST COVER is FPT: if $k < \log n$, the answer for k-TEST COVER is NO and, otherwise, $n \leq 2^k$ and $m \leq 2^n \leq 2^{2^k}$ and so we can solve k-TEST COVER by brute force in time dependent only on k.

(ii) In Section 2, we provide a polynomial-time reduction from the INDEPENDENT SET problem to $(m-k)$-TEST COVER to show that $(m-k)$-TEST COVER is W[1]-hard. A reduction from $(m-k)$-SET COVER to $(m-k)$-TEST COVER and a result from [8] allows us to conclude that $(m-k)$-TEST COVER is W[1]-complete. Thus, $(m-k)$-TEST COVER is not fixed-parameter tractable unless FPT=W[1].

(iii) In Section 3, we prove the main result of this paper: $(n-k)$-TEST COVER is FPT. The proof is quite nontrivial and utilizes a "miniaturized" version of $(n-k)$-TEST COVER introduced and studied in Subsection 3.1.

(iv) Moret and Shapiro [16] obtained a polynomial-time reduction from SET COVER to TEST COVER such that the SET COVER problem has a solution of size k if and only if its reduction to TEST COVER has a solution of size $k + \lceil \log n \rceil$. Since k-SET COVER is W[2]-complete [5], we conclude that $(\log n + k)$-TEST COVER is W[2]-hard. Thus, $(\log n + k)$-TEST COVER is not fixed-parameter tractable provided FPT$\neq$ W[2].

Basics on Parameterized Complexity. A parameterized problem Π can be considered as a set of pairs (I, k), where I is the *problem instance* and k (usually a nonnegative integer) is the *parameter*. Π is called *fixed-parameter tractable* if membership of (I, k) in Π can be decided by an algorithm of runtime $O(f(k)|I|^c)$, where $|I|$ is the size of I, $f(k)$ is an arbitrary function of the parameter k only, and c is a constant independent from k and I. The class of fixed-parameter tractable problems is denoted by FPT.

When the decision time is replaced by the much more powerful $O(|I|^{f(k)})$, we obtain the class XP, where each problem is polynomial-time solvable for any fixed value of k. There is an infinite number of parameterized complexity classes between FPT and XP (for each integer $t \geq 1$, there is a class W[t]) and they form the following tower: $FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq XP$. For the definition of classes W[t], see, e.g., [5,6]. It is well-known that FPT$\neq$XP and it is widely believed that already FPT$\neq$W[1]. Thus, by proving that a problem is W[1]-hard, we essentially rule out that the problem is fixed-parameter tractable (subject to FPT$\neq$W[1]). For more information on parameterized complexity, see monographs [5,6,17].

2 Complexity of $(m-k)$-Test Cover

In this section we give the hardness result for $(m-k)$-Test Cover.

Theorem 1. $(m-k)$-Test Cover *is W[1]-complete.*

Proof. We will give a reduction from the W[1]-hard k-Independent Set problem to $(m-k)$-Test Cover. An input to k-Independent Set consists of an undirected graph $G=(V,E)$ and a positive integer k (the parameter) and the objective is to decide whether there exists an independent set of size at least k in G. A set $I \subseteq V$ is *independent* if no edge of G has both end-vertices in I.

Let G be an input graph to k-Independent Set with vertices $v_1, \ldots, v_p$ and edges $e_1, \ldots, e_q$. We construct an instance of $(m-k)$-Test Cover as follows. The set of items is $\{e_i, e'_i : \ i \in [q]\}$ and the collection of tests is $\{T_j : j \in \ [p]\} \cup \ \{T'_i : i \in [q-1]\}$, where $T_j = \{e_i : \ v_j \in e_i\}$, the set of of edges of G incident to v_j, and $T'_i = \{e_i, e'_i\}$.

A set U of vertices of $G=(V,E)$ is a *vertex cover* if every edge of G has at least one end-vertex in U. It is well-known and easy to see that U is a vertex cover if and only if $V \setminus U$ is an independent set. Consider a minimum size vertex cover U of G, and a test subcollection $\{T_j : v_j \in U\} \cup \{T'_i : i \in [q-1]\}$. Observe that the latter is a test cover, since a pair e_i, e'_j $(i \neq j)$ is separated by $T'_{\min\{i,j\}}$, as are the pairs e_i, e_j and e'_i, e'_j, and a pair e_i, e'_i is separated by T_j for some $v_j \in U$ such that $v_j \in e_i$. Such a v_j exists since U is a vertex cover.

A test cover must use all T'_i as otherwise we cannot separate e'_i, e'_q for some $i \neq q$. A test cover must also use at least $|U|$ of T_j tests. Suppose not, and consider the corresponding set W of vertices, such that $|W| < |U|$. Then every e_i is separated from e'_i by T_j for some $v_j \in W$, and so W forms a vertex cover, contradicting the minimality of U. Hence G has a vertex cover of size t if and only if there is a test cover of size $q-1+t$.

The number of tests is $M = q-1+p$, and so there is a test cover of size $M-k = q-1+p-k$ if and only if G has an independent set with at least k vertices. Since k-Independent Set is W[1]-hard, $(m-k)$-Test Cover is W[1]-hard as well.

To prove that $(m-k)$-Test Cover is in W[1], we will use the following reduction of Test Cover to Set Cover by Moret and Shapiro [16]. Consider an instance of Test Cover with set $[n]$ of items and set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of tests. The corresponding instance of Set Cover has ground set $V = \{(i,j) : 1 \leq i < j \leq n\}$ and set collection $\{S_q : q \in [m]\}$, where $S_q = \{(i,j) \in V : T_q \text{ separates } i,j\}$. Observe that the instance of Test Cover has a test cover of size μ if and only if the corresponding instance of Set Cover has a set cover of size μ. It is proved in [8, Theorem 4] that $(m-k)$-Set Cover is in W[1]. Hence, $(m-k)$-Test Cover is in W[1] as well. This completes the proof. □

3 Complexity of $(n-k)$-Test Cover

In this section we prove that $(n-k)$-Test Cover is fixed-parameter tractable. Towards this we first introduce an equivalence relation on $[n]$.

Given a subcollection $\mathcal{T}' \subseteq \mathcal{T}$, and two items $i, j \in [n]$, $i \neq j$ we write that $i \equiv_{\mathcal{T}'} j$, if i, j is not separated by any tests in $\mathcal{T}'$. Clearly, $\equiv_{\mathcal{T}'}$ is an equivalence relation on $[n]$. Essentially, each equivalence class is a maximal set $C \subseteq [n]$ such that no pair $i, j \in C$ is separated by a test in $\mathcal{T}'$; we say that C is a *class induced by* $\mathcal{T}'$. Observe that $\mathcal{T}'$ is a test cover if and only if each class induced by $\mathcal{T}'$ is a singleton, i.e., there are exactly n classes induced by $\mathcal{T}'$.

3.1 k-Mini Test Cover

To solve $(n-k)$-Test Cover we first introduce a "miniaturized" version of the problem, namely, the k-Mini Test Cover problem. Here, we are given a set $[n]$ of items and a collection $\mathcal{T} = \{T_1, \ldots, T_m\}$ of tests. As with Test Cover, we assume that $\mathcal{T}$ is a test cover. We say that a subcollection $\mathcal{T}' \subseteq \mathcal{T}$ is a *k-mini test cover* if $|\mathcal{T}'| \leq 2k$ and the number of classes induced by $\mathcal{T}'$ is at least $|\mathcal{T}'| + k$. We say a test T *separates* a set S if there exist $i, j \in S$ such that T separates i, j. Our main goal in this subsection is to show that the $(n-k)$-Test Cover problem and the k-Mini Test Cover problem are equivalent. Towards this we first show the following lemma.

Lemma 1. *Suppose that $\mathcal{T}$ is a test cover for $[n]$, $\mathcal{F} \subseteq \mathcal{T}$ and the number of classes induced by $\mathcal{F}$ is at least $|\mathcal{F}| + k$. Then $\mathcal{F}$ can be extended to a test cover of size at most $n-k$. Moreover, if $\mathcal{T}$ contains all singletons, this is possible by adding only singletons.*

Proof. Add tests from $\mathcal{T}$ to $\mathcal{F}$ one by one such that each test increases the number of classes induced by $\mathcal{F}$, until the number of classes is n. This can be done, since if we have less than n classes, there is a class C containing at least two items. For $i, j \in C$ there exists a test T in $\mathcal{T} \setminus \mathcal{F}$ that separates i, j which may be added to $\mathcal{F}$. If we are only permitted to add singletons, then pick $T = \{i\}$. Let $\mathcal{F}'$ be the subcollection produced from $\mathcal{F}$ in this way. Observe that $\mathcal{F}'$ is a test cover. Since $\mathcal{F}$ induces at least $|\mathcal{F}| + k$ classes, we need to add at most $n - (|\mathcal{F}| + k)$ tests to produce $\mathcal{F}'$. Thus $|\mathcal{F}'| \leq n - k$, as required. □

We now define the notion of a *C-test* as follows.

Definition 1. *Let $C \subseteq [n]$. A test $S \in \mathcal{T}$ is a C-test if $C \setminus S \neq \emptyset$ and $S \cap C \neq \emptyset$ (i.e. S separates C). We also define the* local *portion of a C-test S as $L(S) = C \cap S$ and the* global *portion $G(S) = S \setminus C$.*

In order to prove Theorem 2 below we need the following greedy algorithm.

> **Greedy-mini-test**$(\mathcal{T})$**:**
> Start with $\mathcal{F} = \emptyset$. Add two tests T_i, T_j from $\mathcal{T}$ to $\mathcal{F}$ if this will increase the number of classes induced by $\mathcal{F}$ by at least 3. Add a test T_i from $\mathcal{T}$ to $\mathcal{F}$ if this will increase the number of classes induced by $\mathcal{F}$ by at least 2. Stop the construction if we reach $|\mathcal{F}| \geq 2k-2$.

Lemma 2. *If the algorithm Greedy-mini-test produces a set $\mathcal{F}$ with $|\mathcal{F}| \geq 2k-2$, then $\mathcal{F}$ is a k-mini test cover.*

Proof. Observe that throughout Greedy-mini-test we have at least $\lceil \frac{3}{2}|\mathcal{F}| \rceil + 1$ classes, and when $|\mathcal{F}| \geq 2k-2$ then $\lceil \frac{3}{2}|\mathcal{F}| \rceil + 1 \geq |\mathcal{F}| + k$. By construction we note that $|\mathcal{F}| \leq 2k-1 < 2k$, which implies that $\mathcal{F}$ is a k-mini test cover. □

Theorem 2. *Suppose that $\mathcal{T}$ is a test cover for $[n]$. Then $\mathcal{T}$ contains a test cover of size at most $n-k$ if and only if $\mathcal{T}$ contains a k-mini test cover.*

Proof. First suppose that $\mathcal{T}$ contains a k-mini test cover $\mathcal{F}$. Then by Lemma 1, $\mathcal{F}$ can be extended to a test cover of size at most $n-k$.

Conversely, suppose $\mathcal{T}$ contains a test cover $\mathcal{F}'$ of size at most $n-k$. Now use algorithm Greedy-mini-test on $\mathcal{F}'$. If $|\mathcal{F}| \geq 2k-2$, where $\mathcal{F}$ is produced by Greedy-mini-test, then we are done by Lemma 2, so assume that $|\mathcal{F}| < 2k-2$. This implies that the following holds, as otherwise the algorithm wouldn't have terminated when it did.

1. For every test $T_i \in \mathcal{F}' \backslash \mathcal{F}$, T_i does not separate more than one class induced by $\mathcal{F}$.
2. For every class C induced by $\mathcal{F}$, and for every pair T_i, T_j of C-tests in $\mathcal{F}' \backslash \mathcal{F}$, at least one of $(T_i \cap T_j) \cap C$, $(T_i \backslash T_j) \cap C$, $(T_j \backslash T_i) \cap C$ and $C \backslash (T_i \cup T_j)$ is empty.

It can be seen that these properties hold even if we add one extra test from $\mathcal{F}' \backslash \mathcal{F}$ to $\mathcal{F}$.

Therefore if we add t tests from $\mathcal{F}' \backslash \mathcal{F}$, one at a time, this will subdivide a class C into at most $t+1$ classes. Furthermore, since each test separates at most one class, adding t tests from $\mathcal{F}' \backslash \mathcal{F}$ to $\mathcal{F}$ will increase the number of classes induced by $\mathcal{F}$ by at most t. It follows that $\mathcal{F}'$ induces less than $|\mathcal{F}| + k + |\mathcal{F}' \backslash \mathcal{F}| = |\mathcal{F}'| + k \leq n$ classes. But this is a contradiction as $\mathcal{F}'$ is a test cover. □

By Theorem 2 we get the following result, which allows us to concentrate on k-Mini Test Cover in the next subsection.

Corollary 1. *The problem $(n-k)$-*Test Cover *is FPT if and only if k-*Mini Test Cover *is FPT.*

3.2 Main Result

We start with the following easy observation.

Lemma 3. *Let $\mathcal{T}$ be a test cover. Let $\mathcal{T}^*$ be the test cover formed from $\mathcal{T}$ by adding every singleton not already in $\mathcal{T}$. Then $\mathcal{T}^*$ has a k-mini test cover if and only if $\mathcal{T}$ also has a k-mini test cover.*

Proof. Assume $\mathcal{T}^*$ has a k-mini test cover $\mathcal{F}$. Form $\mathcal{F}'$ from $\mathcal{F}$ by removing all singletons. For each singleton removed the number of classes decreases by at most one. Hence, $\mathcal{F}'$ induces at least $|\mathcal{F}'| + k$ classes, and $|\mathcal{F}'| \leq 2k$. Thus, $\mathcal{F}'$ is a k-mini test cover for $\mathcal{T}$. The other direction is immediate since $\mathcal{T} \subseteq \mathcal{T}^*$. □

Due to Lemma 3, *hereafter we assume that every singleton belongs to $\mathcal{T}$.*

We will apply the algorithm Greedy-mini-test to find a collection $\mathcal{F} \subseteq \mathcal{T}$ of tests. If $|\mathcal{F}| \geq 2k - 2$ then we are done by Lemma 2, so for the rest of the arguments we assume that $|\mathcal{F}| < 2k - 2$. By construction, adding any new test to $\mathcal{F}$ increases the number of classes by at most 1 and adding any two new tests to $\mathcal{F}$ increases the number of classes by at most 2. Let the classes created by $\mathcal{F}$ be denoted by $C_1, C_2, \ldots, C_l$. Note that $l \leq 3k - 2$.

Lemma 4. *Any test $S \in \mathcal{T} \setminus \mathcal{F}$ cannot be a C_i-test and a C_j-test for $i \neq j$.*

Proof. For the sake of contradiction, assume such a test S exists. Then adding S to $\mathcal{F}$ will increase the number of classes by at least 2, a contradiction to the definition of $\mathcal{F}$. □

We may assume without loss of generality in the rest of this section that for all C_i-tests, S, we have $|S \cap C_i| \leq |C_i|/2$. Indeed, suppose $|S \cap C_i| > |C_i|/2$. Then we may replace S in $\mathcal{T}$ with the test $S' = [n] \setminus S$. Observe that two items are separated by S' if and only if they are separated by S, and so replacing S with S' produces an equivalent instance. Furthermore, since $|S \cap C_i| > |C_i|/2$ we have that $|S' \cap C_i| \leq |C_i|/2$. Note that Lemma 4 still holds after replacing S with S', since for all $j \neq i$ either $S' \cap C_j = \emptyset$ or $C_j \subseteq S'$.

Lemma 5. *Any two C_i-tests $S, S' \in \mathcal{T}$ have either $L(S) \subseteq L(S')$ or $L(S') \subseteq L(S)$ or $L(S) \cap L(S') = \emptyset$.*

Proof. For the sake of contradiction, assume S, S' do not satisfy this condition. Then $C_i \cap (S \setminus S')$ is non-empty (otherwise, $L(S) \subseteq L(S')$). Similarly, $C_i \cap (S' \setminus S)$ is non-empty. Since $L(S) \cap L(S') \neq \emptyset$, $C_i \cap S \cap S'$ is non-empty. Finally observe that $|L(S) \cup L(S')| = |L(S)| + |L(S')| - |L(S) \cap L(S')| \leq |C_i|/2 + |C_i|/2 - 1 < |C_i|$. Hence $C_i \setminus (S \cup S')$ is non-empty. Adding S and S' to $\mathcal{F}$ divides C_i into four classes: $C_i \cap (S \setminus S')$, $C_i \cap (S' \setminus S)$, $C_i \cap (S \cap S')$ and $C_i \setminus (S \cup S')$. This contradicts the maximality of $\mathcal{F}$. □

An *out-tree* T is an orientation of a tree which has only one vertex of in-degree zero (called the *root*); a vertex of T of out-degree zero is a *leaf*.

We now build an out-tree O_i as follows. The root of the tree, $r \in V(O_i)$ corresponds to the set C_i. Each vertex $v \in V(O_i) \setminus r$ corresponds to a subset, $S_v \subseteq C_i$ such that there exists a C_i-test $S \in \mathcal{T}$ with $L(S) = S_v$. Note that for a pair of vertices $u, v \in V(O_i)$ if $u \neq v$, then $S_u \neq S_v$. Add an arc from v to w in O_i if $S_w \subset S_v$ and there is no u in O_i with $S_w \subset S_u \subset S_v$. By Lemma 5 we note that O_i is indeed an out-tree.

Lemma 6. *Every non-leaf in O_i has out-degree at least two.*

Proof. Let v be a non-leaf in O_i, and note that $|S_v| \geq 2$. Let w be any child of v in O_i. By definition there exists an item in $S_v \setminus S_w$ (as $|S_v| > |S_w|$), say w'. As there is a singleton $\{w'\} \in \mathcal{T}$ there is a path from v to w' in O_i and as $w' \notin S_w$ the path does not use w. Therefore v has at least one other out-neighbour. □

We now define the *signature* of a set $S' \subset C_i$ as follows.

$$Sig(S') = \{G(S) : \; S \in \mathcal{T} \text{ and } L(S) = S'\}$$

Lemma 7. *We have $|\{Sig(S') : \; S' \subset C_i\}| \leq 2^{2^{3k-1}}$.*

Proof. Let $\mathcal{S}_i$ denote all sets, S, with $C_j \cap S = \emptyset$ or $C_j \subseteq S$ for all j and furthermore $C_i \cap S = \emptyset$. Note that $|\mathcal{S}_i| \leq 2^{l-1} \leq 2^{3k-1}$, since $|\{C_1, \ldots, C_l\} \;\setminus\; \{C_i\}| \leq 3k-1$. Note that tests U and V with $L(U) = S' = L(V)$ have $G(U) \neq G(V)$, as $U \neq V$. Observe that all $G(S)$ in $Sig(S')$ belong to $\mathcal{S}_i$ implying that there is at most $2^{|\mathcal{S}_i|} = 2^{2^{3k-1}}$ different choices for a signature. □

Lemma 8. *There exists a function $f_1(k)$ such that either the depth of the tree O_i (i.e. the number of arcs in a longest path out of the root) is at most $f_1(k)$, or in polynomial time, we can find a vertex v in O_i such that if there is a solution to our instance of $(n-k)$*-Test Cover *then there is also a solution that does not use any test S with $L(S) = S_v$.*

Proof. Let $f_1(k) = (32k-1)2^{2^{3k-1}}$. Assume that the depth of the tree O_i is more than $f_1(k)$ and let $p_0 p_1 p_2 \ldots p_a$ be a longest path in O_i (so $a > f_1(k)$). By Lemma 7 and by the choice of $f_1(k)$, there is a sequence $p_{j_1}, p_{j_2}, \ldots, p_{j_{32k}}$, where $1 \leq j_1 < j_2 < \cdots < j_{32k} \leq a$ and all sets corresponding to $p_{j_1}, p_{j_2}, \ldots, p_{j_{32k}}$ have the same signature.

Let S^* be the set corresponding to $p_{j_{16k}}$. We will show that if there is a solution to our instance of $(n-k)$-Test Cover then there is also a solution that does not use any test S with $L(S) = S^*$.

Assume that there is a solution to our instance of $(n-k)$-Test Cover and assume that we pick a solution $\mathcal{T}'$ with as few tests, S, as possible with $L(S) = S^*$. For the sake of contradiction assume that there is at least one test S' in our solution with $L(S') = S^*$. By Theorem 2 there is a k-mini test cover, $\mathcal{F}'$, taken from $\mathcal{T}'$. Initially let $\mathcal{F}'' = \mathcal{F}'$. While there exists a vertex $r \in C_q$ and $r' \in C_p$ $(q \neq p)$ which are not separated by $\mathcal{F}''$ then add any test from $\mathcal{F}$ which separates r and r' to $\mathcal{F}''$ (recall that $\mathcal{F}$ is the test collection found by

Greedy-mini-test). Note that this increases the size of $\mathcal{F}''$ by 1 but also increases the number of classes induced by $\mathcal{F}''$ by at least 1. We continue this process for as long as possible. As $\mathcal{F}'' \subseteq \mathcal{F} \cup \mathcal{F}'$ we note that $|\mathcal{F}''| \leq 2k + 2k = 4k$. Furthermore, by construction, vertices in different C_j's are separated by tests in $\mathcal{F}''$. Also note that the number of classes induced by $\mathcal{F}''$ is at least $|\mathcal{F}''| + k$ (as the number of classes induced by $\mathcal{F}'$ is at least $|\mathcal{F}'| + k$).

For every test, S, in $\mathcal{F}''$ color the vertex in O_i corresponding to $L(S)$ blue. For every vertex, $v \in V(O_i)$, color v red if all paths from v to a leaf in O_i use at least one blue vertex and v is not already colored blue. Finally for every vertex, $w \in V(O_i)$, color w orange if all siblings of w (i.e. vertices with the same in-neighbour as w) are colored blue or red and w is not colored blue or red. We now need the following:

Claim A: The number of colored vertices in O_i is at most $16k - 2$.
Proof of Claim A: As $|\mathcal{F}''| \leq 4k$ we note that the number of blue vertices is at most $4k$. We will now show that the number of red vertices is at most $4k - 1$. Consider the forest obtained from O_i by only keeping arcs out of red vertices. Note that any tree in this forest has all its leaves colored blue and all its internal vertices colored red. Furthermore, by Lemma 6 the out-degree of any internal vertex is at least 2. This implies that the number of red vertices in such a tree is less than the number of blue vertices. As this is true for every tree in the forest we conclude that the number of red vertices in O_i is less than the number of blue vertices in O_i and is therefore bounded by $4k - 1$.

We will now bound the number of orange vertices. Since every orange vertex in O_i has at least one sibling colored blue or red (by Lemma 6). and any blue or red vertex can have at most one orange sibling we note that the number of orange vertices cannot be more than the number of vertices colored blue or red. This implies that the number of orange vertices is at most $8k - 1$.

By Lemma 1, we note that some test, S^x, in $\mathcal{F}''$ has $L(S^x) = S^*$ (as otherwise extend $\mathcal{F}''$ by singletons to a test cover where no test, S, in the solution has $L(S) = S^*$, a contradiction to our assumption). Now create $\mathcal{F}^x$ as follows. Initially let $\mathcal{F}^x$ be obtained from $\mathcal{F}''$ by removing the test S^x. Let $p_{j_{i'}}$ be an uncolored vertex in $\{p_{j_1}, p_{j_2}, \ldots, p_{j_{16k-1}}\}$ and let $p_{j_{i''}}$ be an uncolored vertex in $\{p_{j_{16k+1}}, p_{j_{16k+2}}, \ldots, p_{j_{32k-1}}\}$ (note that we do not pick $p_{j_{32k}}$). Let S^x_1 be a test in $\mathcal{T}$ with $G(S^x_1) = G(S^x)$ and $L(S^x_1)$ corresponding to the vertex $p_{j_{i'}}$ and let S^x_2 be a test in $\mathcal{T}$ with $G(S^x_2) = G(S^x)$ and $L(S^x_2)$ corresponding to $p_{j_{i''}}$. These tests exist as the signature of all sets corresponding to vertices in $p_{j_1}, p_{j_2}, \ldots, p_{j_{32k}}$ are the same. Now add S^x_1 and S^x_2 to $\mathcal{F}^x$. The following now holds.

Claim B: The number of classes induced by $\mathcal{F}^x$ is at least $|\mathcal{F}^x| + k$.
Proof of Claim B: Let $u, v \in [n]$ be arbitrary. If $u, v \notin C_i$ and they are separated by $\mathcal{F}''$, then they are also separated by $\mathcal{F}^x$, as if they were separated by S^x then they will now be separated by S^x_1 (and S^x_2). Now assume that $u \in C_i$ and $v \notin C_i$. If $u \in L(S^x)$ and u and v were separated by S^x then they are also separated by S^x_1. If $u \notin L(S^x)$ and u and v were separated by S^x then they are also separated

by S_2^x. So as u and v were separated by $\mathcal{F}''$ we note that they are also separated by $\mathcal{F}^x$. We will now show that the number of classes completely within C_i using $\mathcal{F}^x$ is at least one larger than when using $\mathcal{F}''$.

By Lemma 5 we note that deleting S^x from $\mathcal{F}''$ can decrease the number of classes within C_i by at most one (it may decrease the number of classes in $[n]$ by more than one). We first show that adding the test S_1^x to $\mathcal{F}'' \setminus \{S^x\}$ increases the number of classes within C_i by at least one. As $p_{j_{i'}}$ is not colored there is a path from $p_{j_{i'}}$ to a leaf, say u_1, without any blue vertices. Furthermore as $p_{j_{i'}}$ is not orange we note that it has a sibling, say s', that is not colored and therefore has a path to a leaf, say u_2, without blue vertices. We now note that u_1 and u_2 are not separated in $\mathcal{F}''$ (and therefore in $\mathcal{F}'' \setminus \{S^x\}$). However adding the test S_1^x to $\mathcal{F}'' \setminus \{S^x\}$ does separate u_1 and u_2 (as $u_1 \in S_1^x$ but $u_2 \notin S_1^x$). Therefore the classes within C_i has increased by at least one by adding S_1^x to $\mathcal{F}'' \setminus \{S^x\}$.

Analogously we show that adding the test S_2^x to $\mathcal{F}'' \cup \{S_1^x\} \setminus \{S^x\}$ increases the number of classes within C_i by at least one. As $p_{j_{i''}}$ is not colored there is a path from $p_{j_{i''}}$ to a leaf, say v_1, without blue vertices. Furthermore as $p_{j_{i''}}$ is not orange we note that it has a sibling, say s'', that is not colored and therefore has a path to a leaf, say v_2, without blue vertices. We now note that v_1 and v_2 are not separated in $\mathcal{F}''$ (and therefore in $\mathcal{F}'' \cup \{S_1^x\} \setminus \{S^x\}$, as $p_{j_{i'}}$ lies higher in the tree O_i and therefore the test S_1^x does not separate u and v). However adding the test S_2^x to $\mathcal{F}'' \cup \{S_1^x\} \setminus \{S^x\}$ does separate v_1 and v_2 (as $v_1 \in S_2^x$ but $v_2 \notin S_2^x$). Therefore the classes within C_i has increased by at least one by adding S_2^x to $\mathcal{F}'' \cup \{S_1^x\} \setminus \{S^x\}$. So we conclude that the number of classes within C_i has increased by at least one and as any vertex not in C_i is still separated from exactly the same vertices in $\mathcal{F}^x$ as it was in $\mathcal{F}''$ we have proved Claim B.

By Lemma 1 and Claim B we get a solution with fewer tests, S, with $L(S) = S^*$, a contradiction. □

Suppose the depth of O_i is greater than $f_1(k)$, and let S^* be the set found by the above lemma. Then we can delete all tests, S, with $L(S) = S^*$ from $\mathcal{T}$ without changing the problem, as if there is a solution for the instance then there is one that does not contain any test S with $L(S) = S^*$. Therefore we may assume that the depth of O_i is at most $f_1(k)$.

Lemma 9. *There exist functions $f_2(d,k)$ and $f_3(d,k)$, such that in polynomial time we can reduce $([n], \mathcal{T}, k)$ to an instance such that the following holds for all vertices $v \in O_i$, where d is the length (i.e. number of arcs) of a longest path out of v in O_i: (1) $N^+(v) \leq f_2(d,k)$ and (2) $|S_v| \leq f_3(d,k)$.*

Proof. Let v be a vertex in O_i and let d be the length of a longest path out of v in O_i. We will prove the lemma by induction on d. If $d = 0$ then v is a leaf in O_i and $N^+(v) = 0$ and $|S_v| = 1$ (as all singletons exist in $\mathcal{T}$). So now assume that $d \geq 1$ and the lemma holds for all smaller values of d. We note that the way we construct $f_3(d,k)$ below implies that it is increasing in d.

We will first prove part (1). Let $N^+(v) = \{w_1, w_2, w_3, \ldots, w_b\}$ and note that $|S_{w_j}| \leq f_3(d-1,k)$ for all $j = 1, 2, \ldots, b$ (by induction and the fact that $f_3(d,k)$

is increasing in d). Let Q_j be the subtree of O_i that is rooted at w_j for all $j = 1, 2, \ldots, b$. As part (1) holds for all vertices in Q_j we note that there are at most $g(d, k)$ non-isomorphic trees in $\{Q_1, Q_2, \ldots, Q_b\}$ for some function $g(d, k)$. Furthermore the number of vertices in each Q_j is bounded by $2f_3(d-1, k) - 1$ by Lemma 6 and induction (using part (2) and the fact that every leaf in Q_j corresponds to a singleton in C_i and the number of leaves are therefore bounded by $f_3(d-1, k)$). By Lemma 7 the number of distinct signatures is bounded by $2^{2^{3k-1}}$. Let $f_2(d, k)$ be defined as follows.

$$f_2(d, k) = 2k \cdot g(d, k) \left[2^{2^{3k-1}}\right]^{2f_3(d-1,k)-1}$$

So if $b > f_2(d, k)$ there exists at least $2k + 1$ trees in $\{Q_1, Q_2, \ldots, Q_b\}$ which are strongly isomorphic, in the sense that a one-to-one mapping from one to the other maintains arcs as well as signatures (a vertex with a given signature is mapped into a vertex with the same signature). Without loss of generality assume that Q_1 is one of these at least $2k + 1$ trees. We now remove all vertices in Q_1 as well as all tests S with $L(S)$ corresponding to a vertex in Q_1. Delete all the items in S_{w_1}. Let the resulting test collection be denoted by $\mathcal{T}'$, and denote the new set of items by $[n']$. We show this reduction is valid in the following claim.

Claim: This reduction is valid (i.e. $([n], \mathcal{T}, k)$ and $([n'], \mathcal{T}', k)$ are equivalent).
Proof of Claim: Observe that any k-mini test cover in $\mathcal{T}'$ is a k-mini test cover in $\mathcal{T}$, and so $([n], \mathcal{T}, k)$ is a YES-instance if $([n'], \mathcal{T}', k)$ is a YES-instance.

For the converse, assume $\mathcal{T}$ contains a k-mini test cover $\mathcal{F}'$, and for each test S in $\mathcal{F}'$, color the vertex in O_i corresponding to $L(S)$ blue. We first show we may assume Q_1 is uncolored. For suppose not, then since $|\mathcal{F}'| \leq 2k$, then some other tree Q_j that is strongly isomorphic to Q_1 is uncolored. In this case, we may replace the tests S in $\mathcal{F}'$ with $L(S)$ corresponding to a vertex in Q_1, by the equivalent tests S' with $L(S')$ corresponding to a vertex in Q_j.

So assume Q_1 is uncolored. Then $\mathcal{F}'$ is still a subcollection in $\mathcal{T}'$. It remains to show that $\mathcal{F}'$ still induces at least $|\mathcal{F}'| + k$ classes over $[n']$. Observe that this holds unless there is some class C induced by $|\mathcal{F}'|$ that only contains items from S_{w_1}. But this can only happen if some item in S_{w_1} is separated from S_{w_j} by a test in $\mathcal{F}'$, for all $j \in \{2, \ldots b\}$. But since $b > |\mathcal{F}| + 1$, there exists $j \neq 1$ such that Q_j is not coloured. Then since w_1, w_j are siblings, no test in $\mathcal{F}'$ can separate S_{w_1} from S_{w_j}. Thus $\mathcal{F}'$ induces at least $|\mathcal{F}'| + k$ classes over $[n']$, and so $\mathcal{F}'$ is still a k-mini test cover in the new instance. Thus, $([n'], \mathcal{T}', k)$ is a YES-instance if and only if $([n], \mathcal{T}, k)$ is a YES-instance.

By the above claim we may assume that $b \leq f_2(d, k)$, which proves part (1). We will now prove part (2). As we have just proved that $b \leq f_2(d, k)$ and $|S_{w_j}| \leq f_3(d-1, k)$ for all $j = 1, 2, \ldots, b$, we note that (2) holds with $f_3(d, k) = f_3(d-1, k) \times f_2(d, k)$. □

Theorem 3. *The* $(n-k)$-TEST COVER *problem is fixed-parameter tractable.*

Proof. Given a test cover, construct a subcollection $\mathcal{F} \subseteq \mathcal{T}$ using algorithm Greedy-mini-test. If $|\mathcal{F}| \geq 2k$, the instance is a Yes-instance since $\mathcal{F}$ induces at least $\frac{3}{2}|\mathcal{F}|$ classes. Otherwise, $|\mathcal{F}| < 2k$ and $\mathcal{F}$ induces at most $3k$ classes. By Lemma 8, we may assume that O_i has depth at most $d = f_1(k)$, and by Lemma 9 part (2) we may assume that $|C_i| \leq f_3(d, k)$, for each class C_i induced by $\mathcal{F}$. Thus $|C_i| \leq f_3(f_1(k), k)$.

Hence there are at most $3k$ classes, the size of each bounded by a function of k, so the number of items in the problem is bounded by a function of k. Thus, the problem can be solved by an algorithm of runtime depending on k only. □

4 Conclusion

We have considered four parameterizations of Test Cover and established their parameterized complexity. The main result is fixed-parameter tractability of $(n-k)$-Test Cover. Whilst it is a positive result, the runtime of the algorithm that we can obtain is not practical and we hope that subsequent improvements of our result can bring down the runtime to a practical level. Ideally, the runtime should be $c^k(n+m)^{O(1)}$, but it is not always possible [13].

Acknowledgment. This research was partially supported by an International Joint grant of Royal Society.

References

1. Alon, N., Gutin, G., Kim, E.J., Szeider, S., Yeo, A.: Solving Max-r-Sat above a tight lower bound. Algorithmica 61, 638–655 (2011)
2. Bondy, J.A.: Induced subsets. J. Combin. Th., Ser. B 12, 201–202 (1972)
3. Crowston, R., Fellows, M., Gutin, G., Jones, M., Rosamond, F., Thomassé, S., Yeo, A.: Simultaneously satisfying linear equations over $\mathbb{F}_2$: MaxLin2 and Max-r-Lin2 parameterized above average. In: Proc. FSTTCS 2011. LIPICS, vol. 13, pp. 229–240 (2011)
4. Crowston, R., Gutin, G., Jones, M., Kim, E.J., Ruzsa, I.Z.: Systems of Linear Equations over $\mathbb{F}_2$ and Problems Parameterized above Average. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 164–175. Springer, Heidelberg (2010)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman and Co. (1979)
8. Gutin, G., Jones, M., Yeo, A.: Kernels for Below-Upper-Bound Parameterizations of the Hitting Set and Directed Dominating Set Problems. Theor. Comput. Sci. 412, 5744–5751 (2011)
9. Gutin, G., Kim, E.J., Szeider, S., Yeo, A.: A probabilistic approach to problems parameterized above or below tight bounds. J. Comput. Syst. Sci. 77, 422–429 (2011)
10. Gutin, G., van Iersel, L., Mnich, M., Yeo, A.: All Ternary Permutation Constraint Satisfaction Problems Parameterized Above Average Have Kernels with Quadratic Number of Variables. J. Comput. Syst. Sci. 78, 151–163 (2012)

11. Halldórsson, B.V., Halldórsson, M.M., Ravi, R.: On the Approximability of the Minimum Test Collection Problem. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 158–169. Springer, Heidelberg (2001)
12. Halldórsson, B.V., Minden, J.S., Ravi, R.: PIER: Proteinidentification by epitope recognition. In: Proc. Currents in Computational Molecular Biology 2001, pp. 109–110 (2001)
13. Lokshtanov, D., Marx, D., Saurabh, S.: Known Algorithms on Graphs on Bounded Treewidth are Probably Optimal. In: Proc. SODA, pp. 777–789 (2011)
14. Mahajan, M., Raman, V.: Parameterizing above guaranteed values: MaxSat and MaxCut. J. Algorithms 31, 335–354 (1999)
15. Mahajan, M., Raman, V., Sikdar, S.: Parameterizing above or below guaranteed values. J. Comput. Syst. Sci. 75, 137–153 (2009)
16. Moret, B.M.E., Shapiro, H.D.: On minimizing a set of tests. SIAM J. Scientific & Statistical Comput. 6, 983–1003 (1985)
17. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Univ. Press (2006)

Sitting Closer to Friends Than Enemies, Revisited

Marek Cygan[1,⋆], Marcin Pilipczuk[2,⋆⋆],
Michał Pilipczuk[3,⋆⋆⋆], and Jakub Onufry Wojtaszczyk[4]

[1] IDSIA, University of Lugano, Switzerland
marek@idsia.ch
[2] Institute of Informatics, University of Warsaw, Poland
malcin@mimuw.edu.pl
[3] Department of Informatics, University of Bergen, Norway
michal.pilipczuk@ii.uib.no
[4] Google Inc., Warsaw, Poland
onufry@google.com

Abstract. Signed graphs, i.e., undirected graphs with edges labelled with a plus or minus sign, are commonly used to model relationships in social networks. Recently, Kermarrec and Thraves [13] initiated the study of the problem of appropriately visualising the network: They asked whether any signed graph can be embedded into the metric space $\mathbb{R}^l$ in such a manner that every vertex is closer to all its friends (neighbours via positive edges) than to all its enemies (neighbours via negative edges). Interestingly, embeddability into $\mathbb{R}^1$ can be expressed as a purely combinatorial problem. In this paper we pursue a deeper study of this case, answering several questions posed by Kermarrec and Thraves.

First, we refine the approach of Kermarrec and Thraves for the case of complete signed graphs by showing that the problem is closely related to the recognition of proper interval graphs. Second, we prove that the general case, whose polynomial-time tractability remained open, is in fact NP-complete. Finally, we provide lower and upper bounds for the time complexity of the general case: we prove that the existence of a subexponential time (in the number of vertices and edges of the input signed graph) algorithm would violate the Exponential Time Hypothesis, whereas a simple dynamic programming approach gives a running time single-exponential in the number of vertices.

1 Introduction

Undirected graphs with edges labelled positively (by a +) and negatively (by a −), called *signed graphs*, in many applications serve as a very simple model

⋆ Partially supported by ERC Starting Grant NEWNET 279352 and Foundation for Polish Science.
⋆⋆ Partially supported by NCN grant N206567140 and Foundation for Polish Science.
⋆⋆⋆ Partially supported by European Research Council (ERC) grant “Rigorous Theory of Preprocessing”, reference 267959.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 296–307, 2012.

of relationships between a group of people, e.g., in a social network. Sign labels can express in a simplified way mutual relations, like staying in a relationship, family bonds or conflicts, by classifying them either as *friendship* (+ edge), *hostility* (− edge) or *ambivalence* (no edge). In particular, much effort has been put into properly understanding and representing the structure of the network, balancing it or naturally partitioning into clusters [1, 3, 5, 14–17, 19]. One of the problems is to visualize the model graph properly, i.e., in such a way that positive relations tend to make vertices be placed close to each other, while negative relations imply large distances between vertices.

In their recent work, Kermarrec and Thraves [13] formalized this problem as follows: Consider the metric space $\mathbb{R}^l$ with the Euclidean metric denoted by d. Given a signed graph G, is it possible to embed the vertices of G in $\mathbb{R}^l$ so that for any positive edge uu_1 and negative edge uu_2 it holds that $d(u, u_1) < d(u, u_2)$? This question has a natural interpretation: we would like to place a group of people so that every person is placed closer to his friends than to his enemies.

The work of Kermarrec and Thraves [13] concentrated on showing a number of examples and counterexamples for embeddability into spaces of small dimensions (1 and 2) and a deeper study of the 1-dimensional case. Interestingly enough, the case of the Euclidean line has an equivalent formulation in the language of pure combinatorics: Given a signed graph G, is it possible to order the vertices of G so that for any positive edge uw there is no negative edge uv with v laying between u and w? The authors made algorithmic use of this combinatorial insight: Providing the given signed graph is complete (i.e., every pair of vertices is adjacent via a positive or negative edge) they show a polynomial-time algorithm that computes an embedding into a line or reports that no such embedding exists.

Kermarrec and Thraves also posed a number of open problems in the area, including the question of the complexity of determining the embeddability of an arbitrary (not necessarily complete) graph into the Euclidean line.

Our Results. We focus on the problem of embedding a signed graph into a line. The reformulation of the 1-dimensional case of Kermarrec and Thraves turns out to be an interesting combinatorial problem, which allows classical methods of analysis and shows interesting links with the class of proper interval graphs.

We begin with refining the result of Kermarrec and Thraves for the case of complete graphs. We prove that a complete signed graph is embeddable into a line if and only if the graph formed by the positive edges is a proper interval graph. Using this theorem one can immediately transfer all the results from the well-studied area of proper interval graphs into our setting. Most importantly, as recognition of proper interval graphs can be performed in linear-time [4], we obtain a simpler algorithm for determining the embeddability of a complete graph into a line, with a linear runtime.

We next analyse the general case. We resolve the open problem posed in [13] negatively: it is NP-complete to resolve whether a given signed graph can be embedded into a line. This hardness result also answers other questions of Kermarrec and Thraves [13]. For example, we infer that it is NP-hard to decide the smallest dimension of a Euclidean space in which the graph can be embedded, as such an algorithm could be used to test embeddability into a line.

Furthermore, we are able to show a lower bound on the time complexity of resolving embeddability into a line, under a plausible complexity assumption. We prove that obtaining an algorithm running in subexponential time (in terms of the total number of vertices and edges of the input graph) would contradict the *Exponential Time Hypothesis* [10] (see Section 2 for an exact statement). We complete the picture of the complexity of the problem by showing a dynamic programming algorithm that runs in $O^\star(2^n)$ time[1], matching the aforementioned lower bound up to a constant in the base of the exponent (n denotes the number of vertices of the input graph).

Organisation of the Paper. In Section 2 we recall widely known notions and facts that are of further use, and provide the details of the combinatorial reformulation of the problem by Kermarrec and Thraves [13]. Section 3 is devoted to refinements in the analysis of the case of the complete signed graphs, while Section 4 describes upper and lower bounds for the complexity of the general case. Finally, in Section 5 we gather conclusions and ideas for further work.

2 Preliminaries

Basic Definitions. For a finite set V, by an *ordering* of V we mean a bijection $\pi : V \to \{1, 2, \ldots, |V|\}$. We sometimes treat an ordering π as a linear order on V and for $u, v \in V$ we write $u \leq_\pi v$ to denote $\pi(u) \leq \pi(v)$. A *lexicographic ordering* imposed by π on pairs of elements from V is an ordering π' of $V \times V$ defined as follows: $(a, b) \leq_{\pi'} (c, d)$ if and only if $a <_\pi c$, or $a = c$ and $b \leq_\pi d$. If π is an ordering of vertices of a directed graph G, then we say that π is a *topological ordering* if and only if for every $(v, w) \in E(G)$ we have that $v \leq_\pi w$. A directed graph admits a topological ordering of vertices if and only if it is acyclic.

In a graph $G = (V, E)$ the *neighbourhood* of a vertex v, denoted $N(v)$, is the set of all its neighbours, i.e., $\{w : vw \in E\}$. The *closed neighbourhood* of v is defined as $N[v] = N(v) \cup \{v\}$.

A *signed graph* is a triple $G = (V, E^+, E^-)$, where $E^+, E^- \subseteq V^{[2]}$ and $E^+ \cap E^- = \emptyset$. We view a signed graph as an undirected simple graph with two possible labels on the edges: positive (+) and negative (−). We call the edges from E^+ positive, while those from E^- — negative. The graph $G^+ = (V, E^+)$ is called the *positive part* of G, and $G^- = (V, E^-)$ — the *negative part*. A signed graph is called *complete* if $E^+ \cup E^- = V^{[2]}$, i.e., every pair of vertices is adjacent via a positive or negative edge.

Proper Interval Graphs. Let $G = (V, E)$ be an undirected graph, $\mathcal{I}$ be a family of size $|V|$ of intervals on real line with nonempty interiors and pairwise different endpoints and $\iota : V \to \mathcal{I}$ be any bijection. We say that $\mathcal{I}$ is an *interval model* for G if for every $v, w \in V$, $v \neq w$, $vw \in E$ is equivalent to $\iota(v) \cap \iota(w) \neq \emptyset$. $\mathcal{I}$ is a

[1] The $O^\star()$ notation suppresses factors that are polynomial in the input size.

proper interval model if, additionally, none of the intervals is entirely contained in any other. Graphs having an interval model are called *interval graphs*, while if a proper interval model exists as well, we call them *proper interval graphs*. We will omit the mapping ι whenever it is clear from the context.

Exponential Time Hypothesis [10]: The *Exponential Time Hypothesis* (ETH for short) asserts that there exists a constant $C > 0$ such that no algorithm solving the 3-CNF-SAT problem in $O(2^{Cn})$ exists, where n denotes the number of variables in the input formula.

Combinatorial Problem Statement. In [13], Kermarrec and Thraves work with the metric definition of the problem: Given a signed graph $G = (V, E^+, E^-)$ a feasible embedding of G in the Euclidean space $\mathbb{R}^l$ is such a function $f : V \to \mathbb{R}^l$ that for all u_1, u_2, u, if $u_1u \in E^+$ and $u_2u \in E^-$, then $d(f(u_1), f(u)) < d(f(u_2), f(u))$ (recall that d stands for the Euclidean distance in $\mathbb{R}^l$). However, for the 1-dimensional case they have in essence proved the following result:

Theorem 1 (Lemmata 3 **and** 4 **of [13], rephrased).** *A signed graph $G = (V, E^+, E^-)$ has a feasible embedding in a line iff there is an ordering π of V such that for every $u \in V$:*

(i) *there are no $u_1 <_\pi u_2 <_\pi u$ such that $u_1u \in E^+$ and $u_2u \in E^-$;*
(ii) *there are no $u_1 >_\pi u_2 >_\pi u$ such that $u_1u \in E^+$ and $u_2u \in E^-$.*

We will jointly call conditions *(i)* and *(ii) the condition imposed on u*. Somewhat abusing the notation, the ordering π will also be called *an embedding of G into the line*. Therefore, from now on we are working with the following combinatorial problem that is equivalent to the version considered by Kermarrec and Thraves:

LINE CLUSTER EMBEDDING
Input: A signed graph $G = (V, E^+, E^-)$.
Task: Does there exist an ordering π on V such that for every $u \in V$: *(i)* there are no $u_1 <_\pi u_2 <_\pi u$ such that $u_1u \in E^+$ and $u_2u \in E^-$; *(ii)* there are no $u_1 >_\pi u_2 >_\pi u$ such that $u_1u \in E^+$ and $u_2u \in E^-$.

3 The Complete Signed Graph Case

In their work, Kermarrec and Thraves [13] announced a polynomial-time algorithm solving the LINE CLUSTER EMBEDDING problem in the case where the input signed graph is complete. Their line of reasoning was essentially as follows: if a signed graph can be embedded into a line, then its positive part has to be chordal. However, for a connected chordal graph with at least 4 vertices that actually is embeddable into a line, every *perfect elimination ordering* of the graph is a feasible solution. Therefore, having checked that the graph is chordal and computed a perfect elimination ordering of every connected component, we can simply verify whether the obtained ordering is a correct line embedding.

We refine the approach of Kermarrec and Thraves by showing that a complete graph has a line embedding if and only if its positive part is a proper interval graph. Recall that proper interval graphs are a subclass of chordal graphs; therefore, the result nicely fits into the picture of Kermarrec and Thraves. Moreover, the theory of proper interval graphs is well-studied, so many results from that area can be immediately translated to our setting. For instance, many NP-complete problems become solvable in polynomial time on proper interval graphs (e.g., [2, 9, 12, 18]), and the linear-time algorithm of Corneil et al. [4] for recognizing proper interval graphs immediately solves the LINE CLUSTER EMBEDDING problem in linear time in case of a complete signed graph.

Theorem 2. *A complete signed graph $G = (V, E^+, E^-)$ is embeddable in $\mathbb{R}^1$ if and only if $G^+ = (V, E^+)$ is a proper interval graph. Moreover, having a feasible ordering π of vertices of V, a proper interval model of G^+ sorted with respect to the left ends of the intervals can be computed in linear time; conversely, having a proper interval model of G^+ sorted with respect to the left ends of the intervals, we can compute a feasible ordering π in linear time.*

Proof. First, let us assume that G^+ is a proper interval graph, and let $\mathcal{I} = \{I_v : v \in V\}$ be a proper interval model of G^+. Notice that as no interval is contained in another, we have a natural order on $\mathcal{I}$ — ordering the intervals with respect to the left ends (or, equivalently, the right ends). We claim that π is a feasible solution for the LINE CLUSTER EMBEDDING instance $G = (V, E^+, E^-)$. Take any $u \in V$. Assume that there were some $u_1 <_\pi u_2 <_\pi u$ such that $u_1u \in E^+$ and $u_2u \in E^-$; this implies intervals I_{u_1} and I_u would overlap. This, in turn, means that the right end of interval I_{u_1} would be on the right of the left end of interval I_u. Therefore, the left and right ends of I_{u_2} are on different sides of the left end of I_u, as $u_2 <_\pi u$ and $u_1 <_\pi u_2$, so I_{u_2} and I_u overlap. This is a contradiction with $u_2u \notin E^+$. A symmetrical argument for the second case finishes the proof in this direction.

Now let us assume that G is embeddable in the line and let π, an ordering of V, be a solution. Moreover, let $v^\leftarrow$ be the first (with respect to π) vertex in the closed neighbourhood of v in G^+, while let $v^\rightarrow$ be the last. Of course, $v^\leftarrow \leq_\pi v \leq_\pi v^\rightarrow$. Let us define a family of intervals on $\mathbb{R}$: let $I_v = \left[\pi(v), \pi(v^\rightarrow) + \frac{\pi(v)}{|V|+1}\right]$ for $v \in V$ and $\mathcal{I} = \{I_v : v \in V\}$. Observe that intervals I_v have nonempty interior and pairwise different endpoints. Now, we prove that (1) no I_v is fully contained in some other I_w and (2) for all $v, w \in V$, $vw \in E^+$ if and only if $I_v \cap I_w \neq \emptyset$. This suffices to show that $\mathcal{I}$ is a proper interval model for G^+.

In order to establish (1), let us assume the contrary: there exists a pair of vertices v, w such that $\pi(v) > \pi(w)$ and $\pi(v^\rightarrow) + \frac{\pi(v)}{|V|+1} < \pi(w^\rightarrow) + \frac{\pi(w)}{|V|+1}$. Then $\pi(v^\rightarrow) < \pi(w^\rightarrow)$. Therefore, by definition of $v^\rightarrow$, $vw^\rightarrow \in E^-$. On the other hand, $ww^\rightarrow \in E^+$ and $w <_\pi v <_\pi w^\rightarrow$, a contradiction with the assumption that π was a proper embedding.

Now we proceed to the proof of (2). Take any two distinct vertices v, w, without losing generality assume that $v <_\pi w$. If $vw \in E^+$, then $\pi(v) < \pi(w)$ and $\pi(w) \leq \pi(v^\rightarrow) < \pi(v^\rightarrow) + \frac{\pi(v)}{|V|+1}$, so I_v and I_w overlap. On the other hand if $vw \in E^-$, then from the condition imposed on v it follows that $w >_\pi v^\to$. Consequently, $\pi(w) > \pi(v^\to)$ and, as $\pi(v) < |V|+1$, also $\pi(w) > \pi(v^\to) + \frac{\pi(v)}{|V|+1}$. Therefore, in this case I_v and I_w do not overlap. This proves $\mathcal{I}$ is in fact a proper interval model for G^+.

The algorithmic part of the theorem statement follows directly from the presented constructions. □

Let us recall the result of Corneil et al. [4], which states that proper interval graphs can be recognized in linear time and the algorithm can also output an ordering of the vertices with respect to the left ends of intervals in some model. We can pipeline this routine with Theorem 2 to obtain the following corollary:

Theorem 3. *Assuming the input graph is complete and given as the set of positive edges,* Line Cluster Embedding *can be solved in $O(|V| + |E^+|)$ time complexity. Moreover, the algorithm can produce a feasible ordering of the vertices in the same time, if such an ordering exists.*

4 The General Case

4.1 *NP*-Completeness of the General Case

In [13] Kermarrec and Thraves asked whether the Line Cluster Embedding problem is also polynomial-time solvable in the case where the input is not restricted to complete graphs. In this section we show that this is unlikely: in fact, the problem becomes NP-complete.

In the proof we use an auxiliary problem, called Acyclic Partition.

Acyclic Partition
Input: A directed graph $D = (V, A)$.
Task: Is it possible to partition V into two sets V_1 and V_2, so that both $D[V_1]$ and $D[V_2]$ are directed acyclic graphs (DAGs)?

Acyclic Partition has been introduced and proven to be NP-complete by Eppstein and Mumford [6] and, independently, by Guillemot et al. [8]. In both these papers it was used as a pivot problem for proving NP-completeness of other problems. For the sake of completeness we would like to revisit the NP-hardness proof, because we need explicit bounds on the size of the directed graph obtained in the reduction for further applications. We begin with an instance of the Set Splitting problem, which is known to be NP-complete [7].

Set Splitting
Input: A set system $(\mathcal{F}, U)$, where $\mathcal{F} \subseteq 2^U$.
Task: Does there exist a subset $X \subseteq U$ such that each set in $\mathcal{F}$ contains both an element from X and an element from $U \setminus X$?

Lemma 4. *There exists a polynomial-time algorithm that given an instance $(\mathcal{F}, U)$ of* Set Splitting *outputs an equivalent instance $G = (V, A)$ of* Acyclic Partition, *for which $|V| = |U| + \sum_{F \in \mathcal{F}} |F|$ and $|A| = 3 \sum_{F \in \mathcal{F}} |F|$.*

Proof. We construct the directed graph $D = (V, A)$ as follows. For every set $F \in \mathcal{F}$ and every $u \in F$ we build a vertex c_u^F and connect all the vertices corresponding to the same set F into a directed cycle in any order. For every element $u \in U$ we build a vertex d_u and for every vertex of the form c_u^F we introduce two arcs: (d_u, c_u^F) and (c_u^F, d_u). This concludes the construction; it is easy to verify the claimed sizes of V and A.

Let us formally prove that the instances are equivalent. Let X be any solution to the $(\mathcal{F}, U)$ instance of Set Splitting. Let $V_1 = \{d_u : u \in X\} \cup \{c_u^F : u \in U \setminus X\}$ and $V_2 = \{d_u : u \in U \setminus X\} \cup \{c_u^F : u \in X\}$. As X splits every set $F \in \mathcal{F}$, none of the cycles formed by vertices c_u^F for fixed F is entirely contained in either V_1 or V_2. Also, for every element u the vertex d_u becomes isolated in the corresponding graph $D[V_i]$, as all his neighbours belong to V_{3-i}. Therefore, both $D[V_1]$ and $D[V_2]$ are collections of isolated vertices and directed paths and (V_1, V_2) is a solution to the Acyclic Partition instance.

In the other direction, let (V_1, V_2) be a solution to the instance of Acyclic Partition. Let $X = \{u : d_u \in V_1\} \subseteq U$; we claim that X is a solution to the instance of Set Splitting. Take any $F \in \mathcal{F}$. As the cycle formed by vertices c_u^F is not entirely contained in any of the graphs $D[V_1], D[V_2]$, there exist some u_1 such that $c_{u_1}^F \in V_1$ and u_2 such that $c_{u_2}^F \in V_2$. As the cycles formed by pairs $\{d_{u_1}, c_{u_1}^F\}$ and $\{d_{u_2}, c_{u_2}^F\}$ are also not entirely contained in $D[V_1]$ nor in $D[V_2]$, $d_{u_1} \in V_2$ and $d_{u_2} \in V_1$. Consequently, $u_1 \in U \setminus X$, $u_2 \in X$ and F is split. □

Lemma 5. *There exists a polynomial-time algorithm that given an instance $D = (V, A)$ of* Acyclic Partition *outputs an equivalent instance $H = (V', E^+, E^-)$ of* Line Cluster Embedding*, such that $|V'| = |V| + |A| + 1$, $|E^+| = 2|A|$ and $|E^-| = |A| + |V|$.*

Proof. We construct the graph H as follows: The set of vertices, V', consists of:

- a *special* vertex s;
- for every $e \in A$, a *checker* vertex c_e;
- for every $v \in V$, an *alignment* vertex a_v.

We construct the edges of the signed graph as follows:

- for every $e \in A$, we introduce a positive edge sc_e;
- for every $v \in V$, we introduce a negative edge sa_v;
- for every arc $(v, w) \in A$, we introduce a positive edge $c_{(v,w)}a_v$ and a negative edge $c_{(v,w)}a_w$.

This concludes the construction; it is easy to verify the claimed sizes of V', E^+, E^-.

Let us prove equivalence of the instances. Let π, an ordering of V', be a solution of the Line Cluster Embedding instance (V', E^+, E^-). As the special vertex s is adjacent via positive edges to all the checker vertices, and via negative edges to all the other, alignment, vertices, in the ordering π the checker vertices together with the special vertex have to form an interval, i.e., a set of consecutive elements with respect to π. Let V_1 be the set of those $v \in V$ for which a_v is to the left of this interval, whereas V_2 is the set of those $v \in V$ for which a_v is to the right of this interval. Formally, $V_1 = \{v \in V : a_v \leq_\pi s\}$ and $V_2 = \{v \in V : a_v \geq_\pi s\}$. We claim that (V_1, V_2) is a feasible solution of the ACYCLIC PARTITION instance (V, A). Consider any arc (v, w) such that $v, w \in V_1$. As $a_v \leq_\pi c_{(v,w)}$, $a_w \leq_\pi c_{(v,w)}$, $c_{(v,w)} a_v \in E^+$ and $c_{(v,w)} a_w \in E^-$, then it follows that $a_w \leq_\pi a_v$. Thus, π has to induce a reverse topological ordering on the vertices of $D[V_1]$ and, therefore, $D[V_1]$ has to be acyclic. Symmetrically, $D[V_2]$ has to be acyclic as well, which concludes the proof of (V_1, V_2) being a feasible solution.

Now take any solution (V_1, V_2) of ACYCLIC PARTITION instance (V, A). Let π_1 be any topological ordering of $D[V_1]$ and π_2 be any topological ordering of $D[V_2]$, by which we mean that if (u, v) is an arc of $D[V_1]$, $\pi_1(u) < \pi_1(v)$, and the same holds for π_2. Let us construct an ordering π of V' as follows:

- first, place all the vertices a_v for $v \in V_1$ in the reverse order induced by π_1;
- then, place all the checker vertices $c_{(v,w)}$ for which $v \in V_1$ and $w \in V_2$, in any order;
- then, place all the checker vertices $c_{(v,w)}$ for which $v, w \in V_1$, in reverse lexicographic order imposed by π_1 on pairs (v, w);
- then, place the special vertex s;
- then, place all the checker vertices $c_{(v,w)}$ for which $v, w \in V_2$, in lexicographic order imposed by π_2 on pairs (v, w);
- then, place all the checker vertices $c_{(v,w)}$ for which $v \in V_2$ and $w \in V_1$, in any order;
- finally, place all the vertices a_v for $v \in V_2$ in the order induced by π_2.

We claim that such π is a feasible solution to LINE CLUSTER EMBEDDING instance (V', E^+, E^-).

Note that the positive neighbours of the special vertex s form an interval, therefore the condition imposed on this vertex is satisfied. Now consider a checker vertex $c_{(v,w)}$. If v, w belong to different sets V_1, V_2, then the only negative neighbour of $c_{(v,w)}$ is the first or the last of his closed neighbourhood with respect to π, thus satisfying the condition imposed on $c_{(v,w)}$. In case when $v, w \in V_1$ or $v, w \in V_2$ this is also true, due to π_1, π_2 being topological orderings of $D[V_1]$, $D[V_2]$ respectively.

Now take any vertex a_v, by symmetry assume $v \in V_1$. We need to prove that the condition imposed on a_v is satisfied as well. The neighbours of v consist of:

1. positive neighbours $c_{(v,v')}$, such that $v' \in V_2$;
2. positive neighbours $c_{(v,v')}$, such that $v' \in V_1$;
3. negative neighbours $c_{(v',v)}$, such that $v' \in V_1$;
4. negative neighbours $c_{(v',v)}$, such that $v' \in V_2$.

We now verify that by the construction of π the neighbours of a_v lie in this very order with respect to π. Clearly, the order in which we placed the checkers in π ensures that the neighbours from (1) are placed before the neighbours from (2) and that the neighbours from (3) are placed before the neighbours from (4). Thus the only non-trivial check is whether the vertices from (2) lie before the vertices from (3). Assume otherwise, that there are some v_1', v_2' such that $(v, v_1') \in A$, $(v_2', v) \in A$, but $c_{(v,v_1')} >_\pi c_{(v_2',v)}$. Then $v_2' <_{\pi_1} v$ as π_1 is a topological ordering of $D[V_1]$, so the pair (v_2', v) is lexicographically smaller than the pair (v, v_1'). Hence $c_{(v,v_1')} >_\pi c_{(v_2',v)}$, a contradiction with the construction of π.

We have verified that for all the vertices the conditions imposed on them are satisfied, so the instances are equivalent. □

The NP-completeness of the Set Splitting problem [7], together with Lemmata 4, 5 and a trivial observation that Line Cluster Embedding is in NP, gives us the following theorem.

Theorem 6. *The* Line Cluster Embedding *problem is NP-complete.*

As mentioned before, the question of finding the smallest dimension of the Euclidean space, into which the given graph can be embedded, clearly generalizes testing embeddability into a line. Therefore, we have the following corollary.

Corollary 7. *It is NP-hard to decide the smallest dimension of the Euclidean space, into which a given signed graph can be embedded.*

4.2 Lower Bound on the Complexity

In this subsection we observe that the presented chain of reductions enables us also to establish a lower bound on the complexity of solving Line Cluster Embedding under ETH. Firstly, let us complete the chain of the reductions.

Lemma 8. *There exists a polynomial-time algorithm that given an instance φ of* 3-CNF-SAT *with n variables and m clauses, outputs an equivalent instance $(U, \mathcal{F})$ of* Set Splitting *with $|U| = 2n + 1$ and $\sum_{F \in \mathcal{F}} |F| = 2n + 4m$.*

Proof. We construct the instance $(U, \mathcal{F})$ as follows. The universe U consists of one special element s and two literals $x, \neg x$ for every variable x of φ. The family $\mathcal{F}$ includes *(a)* for every variable x, a set $F_x = \{x, \neg x\}$; *(b)* for every clause C, a set F_C consisting of s and all the literals in C. It is easy to check the claimed sizes of $U, \mathcal{F}$. We claim that the instance of Set Splitting $(U, \mathcal{F})$ is equivalent to the instance φ of 3-CNF-SAT.

Assume that ψ is a boolean evaluation of variables of φ that satisfies φ. We construct a set $X \subseteq U$ as follows: X consists of all the literals that are true in ψ. Now, every set F_x is split, as exactly one of the literals is true and one is false, whereas every set F_C is split as well, as it contains a true literal, which belongs to X, and the special element s, which does not.

Now assume that $X \subseteq U$ is a solution to the Set Splitting instance $(U, \mathcal{F})$. As taking $U \setminus X$ instead of X also yields a solution, without losing generality we can assume that $s \notin X$. Every set F_x is split by X; therefore, exactly one literal of every variable belongs to X and exactly one does not. Let ψ be a boolean evaluation of variables of φ such that it satisfies all the literals belonging to X. Observe that ψ satisfies φ: for every clause C the set F_C has to be split, so, as $s \notin X$, one of the literals of C belongs to X and, thus, is satisfied by ψ. □

Note that by pipelining Lemmata 8, 4 and 5, we obtain a reduction from 3-CNF-SAT to Line Cluster Embedding, where the output instance has a number of vertices and edges bounded linearly in the number of variables and clauses of the input formula. This observation, together with the key tool used in proving complexity lower bounds under Exponential Time Hypothesis, namely the Sparsification Lemma [11], gives us the following theorem.

Theorem 9. *Unless ETH fails, there is a constant $\delta > 0$ such that there is no algorithm that given a (V, E^+, E^-) instance of* Line Cluster Embedding *problem, solves it in $O(2^{\delta(|V|+|E^+|+|E^-|)})$ time.*

Proof. Let us begin by recalling the *Sparsification Lemma.*

Lemma 10 (Sparsification Lemma, Corollary 1 of [11]). *For all $\varepsilon > 0$ and positive k, there is a constant C so that any k-SAT formula Φ with n variables can be expressed as $\Phi = \bigvee_{i=1}^{t} \Psi_i$, where $t \leq 2^{\varepsilon n}$ and each Ψ_i is a k-SAT formula with at most Cn clauses. Moreover, this disjunction can be computed by an algorithm running in time $O^\star(2^{\varepsilon n})$.*

Let us assume that for all $\delta > 0$ there exists an algorithm solving Line Cluster Embedding in $O(2^{\delta(|V|+|E^+|+|E^-|)})$ time complexity. We show an algorithm solving 3-CNF-SAT in $O^\star(2^{\varepsilon n})$ time for every $\varepsilon > 0$, where n is the number of variables, thus contradicting the ETH. Indeed, having fixed ε we can:

- take an instance of 3-CNF-SAT and using Sparsification Lemma in $O^\star(2^{\varepsilon n/2})$ time express it as a disjunction of at most $2^{\varepsilon n/2}$ 3-CNF-SAT instances, each containing at most Cn clauses for some constant C;
- reduce each instance in polynomial time via Set Splitting and Acyclic Partition to Line Cluster Embedding, thus obtaining at most $2^{\varepsilon n/2}$ instances of Line Cluster Embedding, each having $|V|, |E^+|, |E^-| \leq C'n$ for some constant C';
- in each of the instances run the assumed algorithm for Line Cluster Embedding, running in $O(2^{\delta(|V|+|E^+|+|E^-|)})$ time, for $\delta = \frac{\varepsilon}{6C'}$. □

4.3 A Single-Exponential Algorithm for Line Cluster Embedding

Note that the trivial brute-force algorithm for Line Cluster Embedding checks all possible orderings, working in $O^\star(n!)$ time. To complete the picture of the complexity of Line Cluster Embedding, we show that a simple dynamic programming approach can give single-exponential time complexity. This matches the lower bound obtained from under Exponential Time Hypothesis (up to a constant in the base of the exponent).

Before we proceed with the description of the algorithm, let us state a combinatorial observation that will be its main ingredient. Let (V, E^+, E^-) be the given Line Cluster Embedding instance. For $X \subseteq V$ and $v \notin X$ we will say that v is *good* for the set X iff

- no vertex $w \in X$ that is adjacent to v via a negative edge is simultaneously adjacent to some vertex from $V \setminus (X \cup \{v\})$ via a positive edge;
- no vertex $w \in V \setminus (X \cup \{v\})$ that is adjacent to v via a negative edge is simultaneously adjacent to some vertex from X via a positive edge.

Lemma 11. *An ordering π is a feasible solution of (V, E^+, E^-) if and only if every vertex $v \in V$ is good for the set $\{u : u <_\pi v\}$.*

Proof. One direction is obvious: if π is a feasible solution, then every vertex v has to be good for the set $\{u : u <_\pi v\}$. If v would not be good for $\{u : u <_\pi v\}$, there would exist a vertex w certifying that v is not good, and the condition imposed upon w would be not satisfied.

Now assume that every vertex $v \in V$ is good for $\{u : u <_\pi v\}$ and take an arbitrary vertex $v \in V$. If there were vertices $u_1 <_\pi u_2 <_\pi v$ such that $u_1 v \in E^+$ while $u_2 v \in E^-$, then u_2 would not be good for the set $\{u : u <_\pi u_2\}$, a contradiction. Similarly, if there were vertices $u_1 >_\pi u_2 >_\pi v$ such that $u_1 v \in E^+$ while $u_2 v \in E^-$, then u_2 would not be good for the set $\{u : u <_\pi u_2\}$, a contradiction as well. Therefore, the condition imposed on v is satisfied for an arbitrary choice of v. □

Theorem 12. Line Cluster Embedding *can be solved in $O^\star(2^n)$ time and space complexity. Moreover, the algorithm can also output a feasible ordering of the vertices, if it exists.*

Proof. Let (V, E^+, E^-) be the given Line Cluster Embedding instance. Let $W = \{(v, X) : v \text{ is good for } X\}$. Let us construct a directed graph $D = (W, F)$, where $((v, X), (v', X')) \in F$ if and only if $X' = X \cup \{v\}$. As recognizing being good is clearly a polynomial time operation, the graph D can be constructed in $O^\star(2^n)$ time and has that many vertices and edges. Observe that by Lemma 11 there is a feasible ordering π if and only if some sink $(v, V \setminus \{v\})$ is reachable from some source $(u, \emptyset)$; indeed, such a path corresponds to introducing the vertices of V one by one in such a manner that each of them is good for the respective prefix. Reachability of any sink from any source can be, however, tested in time linear in the size of the graph using a breadth-first search. The search can also reconstruct the path in the same complexity, thus constructing the feasible solution. □

5 Conclusions

In this paper we addressed a number of problems raised by Kermarrec and Thraves in [13] for embeddability of a signed graph into a line. We refined their study of the case of a complete signed graph by showing relation with proper interval graphs. Moreover, we have proven NP-hardness of the general case and shown an almost complete picture of its complexity.

Although the general case of the problem appears to be hard, real-life social networks have a certain structure. Is it possible to develop faster, maybe even polynomial-time algorithms for classes of graphs reflecting this structure?

Acknowledgments. We thank Sylvain Guillemot for pointing us to [6] and [8].

References

1. Antal, T., Krapivsky, P.L., Redner, S.: Dynamics of social balance on networks. Phys. Rev. E 72(3), 036121 (2005)
2. Belmonte, R., Vatshelle, M.: Graph Classes with Structured Neighborhoods and Algorithmic Applications. In: Kolman, P., Kratochvíl, J. (eds.) WG 2011. LNCS, vol. 6986, pp. 47–58. Springer, Heidelberg (2011)
3. Cartwright, D., Harary, F.: Structural balance: a generalization of heider's theory. Psychological Review 63(5), 277–293 (1956)
4. Corneil, D.G., Kim, H., Natarajan, S., Olariu, S., Sprague, A.P.: Simple linear time recognition of unit interval graphs. Inf. Process. Lett. 55(2), 99–104 (1995)
5. Davis, J.A.: Clustering and structural balance in graphs. Human Relations 20(2), 181 (1967)
6. Eppstein, D., Mumford, E.: Self-overlapping curves revisited. In: Mathieu, C. (ed.) SODA, pp. 160–169. SIAM (2009)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
8. Guillemot, S., Jansson, J., Sung, W.K.: Computing a smallest multilabeled phylogenetic tree from rooted triplets. IEEE/ACM Trans. Comput. Biology Bioinform. 8(4), 1141–1147 (2011)
9. Ibarra, L.: A simple algorithm to find hamiltonian cycles in proper interval graphs. Inf. Process. Lett. 109(18), 1105–1108 (2009)
10. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. 62(2), 367–375 (2001)
11. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. 63(4), 512–530 (2001)
12. Ioannidou, K., Mertzios, G.B., Nikolopoulos, S.D.: The Longest Path Problem Is Polynomial on Interval Graphs. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 403–414. Springer, Heidelberg (2009)
13. Kermarrec, A.M., Thraves, C.: Can Everybody Sit Closer to Their Friends Than Their Enemies? In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 388–399. Springer, Heidelberg (2011)
14. Kunegis, J., Schmidt, S., Lommatzsch, A., Lerner, J., Luca, E.W.D., Albayrak, S.: Spectral analysis of signed graphs for clustering, prediction and visualization. In: SDM, p. 559–570. SIAM (2010)
15. Leskovec, J., Huttenlocher, D.P., Kleinberg, J.M.: Governance in social media: A case study of the wikipedia promotion process. In: Cohen, W.W., Gosling, S. (eds.) ICWSM. The AAAI Press (2010)
16. Leskovec, J., Huttenlocher, D.P., Kleinberg, J.M.: Predicting positive and negative links in online social networks. In: Rappa, M., Jones, P., Freire, J., Chakrabarti, S. (eds.) WWW, pp. 641–650. ACM (2010)
17. Leskovec, J., Huttenlocher, D.P., Kleinberg, J.M.: Signed networks in social media. In: Mynatt, E.D., Schoner, D., Fitzpatrick, G., Hudson, S.E., Edwards, W.K., Rodden, T. (eds.) CHI, pp. 1361–1370. ACM (2010)
18. Mertzios, G.B.: A polynomial algorithm for the k-cluster problem on the interval graphs. Electronic Notes in Discrete Mathematics 26, 111–118 (2006)
19. Szell, M., Lambiotte, R., Thurner, S.: Multirelational organization of large-scale social networks in an online world. PNAS 107(31), 13636–13641 (2010)

A Dichotomy Theorem for Homomorphism Polynomials

Nicolas de Rugy-Altherre

Univ Paris Diderot, Sorbonne Paris Cité, Institut de Mathématiques de Jussieu,
UMR 7586 CNRS,
F-75205 Paris, France
nderugy@math.univ-paris-diderot.fr

Abstract. In the present paper we show a dichotomy theorem for the complexity of polynomial evaluation. We associate to each graph H a polynomial that encodes all graphs of a fixed size homomorphic to H. We show that this family is computable by arithmetic circuits in constant depth if H has a loop or no edges and that it is hard otherwise (i.e., complete for VNP, the arithmetic class related to $\#P$). We also demonstrate the hardness over $\mathbb{Q}$ of cut eliminator, a polynomial defined by Bürgisser which is known to be neither VP nor VNP-complete in $\mathbb{F}_2$, if VP ≠ VNP (VP is the class of polynomials computable by arithmetic circuits of polynomial size).

1 Introduction

Dichotomy theorems are a way to classify entire classes of problems by their complexity. For instance, Schaefer [Sch78] has classified a subclass of CSP as being in P or NP-complete. Whether such a theorem holds for every CSP is a famous open question, the Dichotomy Conjecture [FV98]. More recently, dichotomy theorems for counting homomorphism problems have been studied by several authors [DG00, BG04, GT11]. In Valiant's theory of polynomial evaluation, an equivalent of Schaefer's theorem has been produced by Briquel and Koiran [BK09], opening the way for such theorems in this theory.

In this paper, we define the class of *homomorphism polynomials*: if H is a graph, the monomials of the associated homomorphism polynomial f_n^H encode the list of graphs of size n homomorphic to H. Our main result is a dichotomy theorem for this class of polynomials, in Valiant's theory: (f_n^H) is computable by unbounded fan-in arithmetic circuits in constant depth (the Valiant equivalent of AC_0) if H has a loop or no edge and hard otherwise (i.e., complete for VNP, the arithmetic class related to $\#P$).

Deciding the existence of or counting homomorphisms are important problems which generalize many natural questions (k-coloring, acyclicity, binary CSP, etc [HN04]). The weighted version, which is very similar to our polynomials, has several applications in statistical physics [Wel93].

The connection between Valiant's theory of polynomial evaluation and counting problems is well known. For instance, the permanent is complete in both

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 308–322, 2012.

settings [Val79a, Val79b]. Homomorphism polynomials are therefore linked to counting homomorphism problems: the evaluation of f_n^H counts the number of graphs of size n homomorphic to H. But these polynomials also tell us something about enumeration: it is equivalent to enumerate this set of graphs and to enumerate all the monomials of f_n^H. We will use this property to demonstrate a result on representations by first order formulas of the bipartite property for graphs.

The hard part of our dichotomy theorem is to demonstrate the hardness of most homomorphism polynomials. We therefore obtain many new VNP-complete families (indeed unlike NP-complete problems, there are not a lot of VNP-complete polynomials known). These families can be seen as generating functions in the way introduced by Jerrum and Bürgisser [Bür00]. No general hardness theorem has been found for these functions (but there are large results for tractable functions [CMR01]). As stated before, many problems can be seen as homomorphism problems. Our theorem therefore implies the hardness of many generating functions.

Moreover, Bürgisser [Bür00] gives a family of polynomials, Cut^2, which is neither VNP-complete nor VP over $\mathbb{F}_2$, if VNP ≠ VP of course. He also demonstrates the existence of such a family in every field, but does not give a specific family for $\mathbb{Q}$. He wonders (Problem 5.2 in [Bür00]) whether Cut^2 can be such a family in $\mathbb{Q}$. During the demonstration of our theorem, we will show the VNP-completeness of Cut^2 and therefore answer negatively the question.

2 Definitions and General Discussions

A polynomial enumerates a family $\mathcal{P}$ of graphs if each monomial of this polynomial encodes a single graph G in $\mathcal{P}$ and each graph $G \in \mathcal{P}$ is encoded by a single monomial. In this paper we study polynomials enumerating graphs G homomorphic to a graph H using a simple code: the variables of the polynomial are the set $\{x_e, e \in E(K_n)\}$; the monomial coding G is $\prod_{e\in E(G)} x_e$. These polynomials are therefore multilinear (i.e., every variable has a maximal degree of 1) and have only 0 or 1 for coefficients. More precisely

Definition 1. Let H be a graph. The polynomial enumerating all graphs G with n vertices which are homomorphic to H is:

$$f_n^H((x_e)_{e\in E(K_n)}) := \sum_{\bar{\epsilon}\in\{0,1\}^{|E(K_n)|}} \Phi_n^H(\bar{\epsilon}) \prod_{e\in E(K_n)} x_e^{\epsilon_e}$$

Where $\Phi_n^H(\bar{\epsilon})$ is equal to 1 if the graph G such that $V(G) = [n]$ and $E(G) = \{e \in E(K_n)|\epsilon_e = 1\}$ is homomorphic to H, 0 otherwise.

We work within Valiant's algebraic framework. Here is a brief introduction to this complexity theory. For a more complete overview, see [Bür00].

An *arithmetic circuit* over a field $\mathbb{K}$ is a labeled directed acyclic connected graph with vertices of indegree 0 or 2 and only one sink. The vertices with indegree 0 are called *input gates* and are labeled with variables or constants from $\mathbb{K}$. The vertices with indegree 2 are called *computation gates* and are labeled with $\times$ or $+$. The sink of the circuit is called the *output gate*.

The polynomial computed by a gate of the arithmetic circuit is defined by induction : an input gate computes its label; a computation gate computes the product or the sum of its children's' values. The polynomial computed by an arithmetic circuit is the polynomial computed by the sink of the circuit.

A *p-family* is a sequence (f_n) of polynomials over a field $\mathbb{K}$ such that the number of variables as well as the degree of f_n are polynomially bounded in n. The *complexity* $L(f)$ of a polynomial $f \in \mathbb{K}[x_1, \ldots, x_n]$ is the minimal number of computational gates of an arithmetic circuit computing f from variables $x_1, \ldots, x_n$ and constants in $\mathbb{K}$.

Two of the main classes in this theory are: the analog of P, VP, which contains every p-family (f_n) such that $L(f_n)$ is a function polynomially bounded in n; and the analog of NP, VNP. A p-family (f_n) is in VNP iff there exists a VP family (g_n) such that for all n, $f_n(x_1, \ldots, x_n) = \sum_{\bar{\epsilon} \in \{0,1\}^n} g_n(x_1, \ldots, x_n, \epsilon_1, \ldots, \epsilon_n)$.

Furthermore we introduce the Valiant equivalent of AC_0, VAC_0 as the class of every p-family (f_n) such that they can be computed by a circuit of size polynomially bounded in n and of constant depth, but whose computational gates have unfounded fan-in. This class has been introduced in [MRar].

As in most complexity theory we have a natural notion of reduction: a polynomial f is a projection of a polynomial g, write $f \leq_p g$, if there are values $a_i \in \mathbb{K} \cup \{x_1, \ldots, x_n\}$ such that $f(\bar{x}) = g(\bar{a})$. A p-family (f_n) is a p-projection of (g_n), also write $(f_n) \leq_p (g_n)$, if there exists a polynomially bounded function p such that for every n, $f_n \leq_p g_{p(n)}$. As usual we say that (g_n) is complete for a class C if, for every $(f_n) \in C$, $(f_n) \leq_p (g_n)$ and if $(g_n) \in C$.

The three classes presented above are closed under projections. We will need in this paper a second notion of reduction, the c-reduction: the oracle complexity $L^g(f)$ of a polynomial f with the knowledge of g is the minimum number of computation gates and evaluations of g over previously computed values that are sufficient to compute f from the variables $x_1, \ldots, x_n$ and constants from $\mathbb{K}$. A p-family (f_n) *c-reduces* to (g_n), write $(f_n) \leq_c (g_n)$, if there exists a polynomially bounded function p such that $L^{g_{p(n)}}(f_n)$ is a polynomially bounded function.

This reduction is more powerful than the projection: if $(f_n) \leq_p (g_n)$, then $(f_n) \leq_c (g_n)$. With this reduction, VNP is still closed, but it is harder to demonstrate (See [Poi08] for an idea of the proof). However this reduction does not distinguish the lower classes. For example, 0 is VP-complete under c-reductions.

The notion of c-reductions was introduced by Bürgisser [Bür00] in his work on p-families that are neither in VP nor VNP-complete (if VNP $\neq$ VP). It has rarely been used to demonstrate the hardness of computing polynomials, though there is a recent example in [Bri11].

Bürgisser demonstrates the existence of such a family in every field with an abstract embedding theorem. Furthermore he gives a specific family for some finite fields, the *cut enumerator* Cut_n^q:

$$\mathrm{Cut}_n^q := \sum_S \prod_{i \in A, j \in B} x_{ij}^{q-1}$$

where the sum is over all cuts $S = \{A, B\}$ of the complete graph K_n on the set of nodes $\{1, \ldots, n\}$. (A cut of a graph is a partition of its set of nodes into two nonempty subsets).

Theorem 1. *(Bürgisser, 5.22) Let q be a power of the prime p. The family of cut enumerators Cut^q over a finite field $\mathbb{F}_q$ is neither* VP *nor* VNP*-complete with respect to c-reductions, provided Mod_p*NP $\not\subseteq$ P*/poly. The latter condition is satisfied if the polynomial hierarchy does not collapse at the second level.*

However, the cut enumerator can be rewritten in the following way:

$$\mathrm{Cut}_n^q(\bar{x}) = \sum_{\substack{V \subsetneq [n] \\ V \neq \emptyset}} \prod_{i \in V} \prod_{j \in V^c} x_{i,j}^{q-1}$$

This polynomial will be studied in Lemma 7, for other purposes. In particular, we will demonstrate that this family is VNP-complete in $\mathbb{Q}$ if $q = 2$, which answers a question of Bürgisser. Furthermore, this implies that our demonstration does not work in a field of characteristic 2 (if VP ≠ VNP).

The main theorem of our paper is:

Theorem 2. *Let H be a graph. Let f_n^H be the polynomial enumerating all graphs homomorphic to H. Then, over $\mathbb{Q}$*

- *If H has a loop or no edges, (f_n^H) is in* VAC_0.
- *Else (f_n^H) is VNP-complete under c-reductions.*

Proofs that all our p-families are indeed p-families and are in VNP are left to the reader. It is a simple application of Valiant's criterion [Bür00]. We will demonstrate our main theorem in four steps: first we will deal with the easy cases. Secondly, using Dyer and Greenhill's ideas, we will reduce the problem to the case where H is as simple as possible, i.e., with only one edge and two vertices, $\bullet\!\!-\!\!\bullet$. In a third step, we will note that the monomials of $f^{\bullet\!-\!\bullet}$ have a kind of "hereditary" property which we will use to reduce the problem to a simpler polynomial. Finally, we will demonstrate the VNP-completeness of this simpler polynomial.

As an application of our theorem and to illustrate the relation with enumeration, here is a sketch of the following corollary. it is more of a proof concept than a fully demonstrated result:

Corollary 1. *If* VP ≠ VNP*, and if the product of two boolean matrices of size n cannot be computed in $\mathcal{O}(n^2)$, being bipartite cannot be expressed by an acyclic conjunctive first order formula of polynomial size on the size of the studied graph.*

We just give here an idea of the demonstration. Let ϕ_n be an acylic conjunctive first order formula of polynomial size stating that G is bipartite or not, for every graph G of size n. As the set of bipartite graphs can be enumerated with a constant delay, by the dichotomy theorem of Bagan [Bag09] (true if the product of two boolean matrices of size n cannot be computed in $\mathcal{O}(n^2)$), ϕ_n is CCQ, i.e., of star size 1. Therefore, thanks to the work of Durand and Mengel [DM11], we know that $P(\phi) \in$ VP. At last, as $P(\phi) = f^{\bullet\!-\!\bullet}$, $f^{\bullet\!-\!\bullet}$ is VNP-complete and in VP, which is impossible if VNP $\neq$ VP. Therefore ϕ_n cannot exist.

3 The Easy Cases

Lemma 1. *If H has a loop, then (f_n^H) is in* VAC_0*.*

Proof. Let v be a looped vertex of H. Then every graph G is homomorphic to H: we just have to send all vertices of G to v. Therefore,

$$f_n^H(\bar{x}) = \sum_{\bar{\epsilon}\in\{0,1\}^{|E(K_n)|}} \prod_{e\in E(K_n)} x_e^{\epsilon_n} = \prod_{e\in E(K_n)} (1 + x_e)$$

This product can be computed easily by an arithmetic circuit of polynomial size, depth 3 and unbounded fan-in for the ×-gate. □

The case where H has no edges is quite obvious, we just have to notice that if H_1 and H_2 are two bihomomorphic graphs, i.e., there exists a homomorphism from H_1 to H_2 and vice versa, then, for all n, $f_n^{H_1} = f_n^{H_2}$. Let us consider now our graph H with no edges. It is bihomomorphic to the graph H_0, composed of a single vertex. Furthermore the only graphs homomorphic to H_0 are those with no edges. Therefore $f_n^{H_0} = 1$.

Let us quote a very useful technical result, already pointed out in [Bür00].

Lemma 2. *Let (f_n) be a p-family. Let us write $\boldsymbol{HC}_k(f_n)$ for the homogeneous component of degree k of f_n, i.e., the sum of every monomial of degree k.*

Then for any sequence of integers (k_n) there exists a c-reduction from the homogeneous component to the polynomial itself:

$$(\boldsymbol{HC}_{k_n}(f_n)) \leq_c (f_n)$$

Proof. Let n be an integer and d_n be the degree of f_n. Let us recall that the degree of f_n is polynomially bounded in n. To begin with, let us notice that for all i, $f_n(2^i\bar{x}) = \sum_{k=0}^{d_n}(2^i)^k\mathbf{HC}_k(f_n)(\bar{x})$. If we write $c_k = \mathbf{HC}_k(f_n)(\bar{x})$, $f_i = f_n(2^i\bar{x})$ and $w_k = 2^k$, those equations can be written as:

$$\begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{d_n} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \omega_0 & \omega_1 & \cdots & \omega_{d_n} \\ \vdots & \vdots & \vdots & \vdots \\ \omega_0^{d_n} & \omega_1^{d_n} & \cdots & \omega_{d_n}^{d_n} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{d_n} \end{pmatrix}$$

Let us write M the square matrix of this equation. It is a Vandermonde matrix with non negative integers. As the coefficients w_k are distinct, this matrix is invertible. Therefore there exist some rationals $w_{i,k}^*$ such that $c_k = \sum_{i=0}^{d_n} w_{i,k}^* f_n(2^i\bar{x})$.

Therefore we have, for all n and all $k \leq d_n$, $L^{f_n}(\mathbf{HC}_k(f_n)) = \mathcal{O}(n \times d_n)$. Let (k_n) be a sequence of integers. If $k_n > d_n$, then $\mathbf{HC}_{k_n}(f_n) = 0$. Thus for all $L^{f_n}(\mathbf{HC}_{k_n}(f_n)) \leq \mathcal{O}(n \times d_n)$. As d_n is polynomially bounded by n, we have just constructed the c-reduction we were looking for. □

This lemma is still true if we take the homogeneous component in some variables only. Let $f(\bar{x}, \bar{y})$ be a polynomial and $\mathbf{HC}_k^{\bar{y}}(f)$ the homogeneous component of degree k in variables $\bar{y}$, i.e., considering $\bar{x}$ as constant. Then for any p-family (f_n) and any sequence of integers k_n,$\left(\mathbf{HC}_{k_n}^{\bar{y}}(f_n)\right) \leq_c (f_n)$.

4 Reduction to Bipartite Graphs

One of the main ideas of Dyer and Greenhill is to reduce the problem of counting homomorphisms from G to H to a simpler graph H. Here we have a similar result but the graph we obtain is extremely simple.

Proposition 1. *Let $\bullet\!\!-\!\!\bullet$ be the graph with two vertices and one edge. Then for any graph H with no loops and at least one edge:*

$$(f_n^{\bullet\!-\!\bullet}) \leq_c \left(f_n^H\right)$$

Lemma 3. *Let $H = (V, E)$ be a graph with no loops and at least one edge. For all $i \in V$, let us call H_i the subgraph of H induced by the neighbours of i and $\tilde{H}$ the disjoint union of all those neighbourhood graphs. Then*

$$\left(f_n^{\tilde{H}}\right) \leq_c \left(f_n^H\right)$$

Proof. let us notice that, as H has no loops, $i \notin H_i$. For any graph G, let us call G' the graph built from G by adding a new vertex v and joining it to every vertex of G. Dyer and Greenhill have noticed in their paper [DG00] that the number of homomorphisms from G' to H is equal to the number of homomorphisms from G to $\tilde{H}$. Let us see what this means for our polynomials.

Let $\tilde{f}_n^H(\bar{x})$ be the polynomial built from f_{n+1}^H by replacing all $x_{i,n+1}$ variables by y, by taking the homogeneous component of degree n in y and by giving value 1 to y: $\tilde{f}_n^H(\bar{x}) = \mathbf{HC}_n^y\left(f_{n+1}^H(\bar{x}, y)\right)_{|y=1}$.

$\tilde{f}_n^H$ enumerates all graphs of size $n+1$ homomorphic to H and having a vertex v linked to all others, i.e., graphs G' homomorphic to H.

$$\tilde{f}_n^H(\bar{x}) = \sum_{G \text{ graph of size } n} \Phi_n^H(G')\bar{x}^{\bar{G}} = \sum_{G \text{ graph of size } n} \Phi_n^{\tilde{H}}(G)\bar{x}^{\bar{G}} := f_n^{\tilde{H}}(\bar{x})$$

where $\Phi_n^H(G) = 1$ if there exists a homomorphism between G and H, 0 otherwise. Then $\tilde{f}_n^H = f_n^{\tilde{H}}$ and consequently there is a c-reduction from $(f_n^{\tilde{H}})$ to (f_{n+1}^H). □

Proof of the Proposition 1. Since H has no loops, the degree of $\tilde{H}$ is strictly lower than that of H. Indeed, for any vertex j of H, any vertex k linked to j in

H' is also linked to j in H. So the degree of $\tilde{H}$ cannot grow. Furthermore, if i is a maximal degree (d) vertex of H and if j is a vertex of H, then either i is not linked to j and so does not appear in H_j, or it is linked to j and, in H_j, it cannot be linked to more than $d-1$ vertices ($|V_i \cap V_j|$, knowing that $j \notin V_j$). The degree of the vertices of $\tilde{H}$ corresponding to the vertex i is therefore strictly lower than d.

If we apply the process of Lemma 3 to H a finite number of times we obtain a graph of maximal degree 1. This graph is bihomomorphic to •–•. Therefore we have the expected reduction: $(f_n^{\bullet\!-\!\bullet}) \leq_c \left(f_n^H\right)$ □

5 The Bipartite Case

Now we have to demonstrate the following proposition.

Proposition 2. *Let •–• be the graph with two vertices and one edge. Then $f_n^{\bullet\!-\!\bullet}$ is VNP-complete.*

5.1 Hereditary Polynomials

$f_n^{\bullet\!-\!\bullet}$ has a noteworthy property: if $(x_i)_{i\in I}$ is a monomial of $f_n^{\bullet\!-\!\bullet}$, then for all $J \subset I$, $(x_j)_{j\in J}$ is also a monomial of $f_n^{\bullet\!-\!\bullet}$. Indeed, this polynomial enumerates all graphs homomorphic to •–•, i.e., all bipartite graphs, and every subgraph of a graph homomorphic to •–• is also homomorphic to •–•. To build on this idea, we introduce the following definitions.

Let f be a multilinear polynomial with 0,1 coefficients only. f is *hereditary* if it satisfies the following property: if $(x_i)_{i\in I}$ is a monomial of f, so is $(x_j)_{j\in J}$, for all $J \subset I$. This relation induces an order on all the monomials of f: $(x_i)_{i\in I}$ is greater that $(x_j)_{j\in J}$ if $J \subset I$. We call a monomial a *generator* for f if it is maximal for this order.

A hereditary polynomial is completely defined by its generators. We can naturally ask the opposite question. Let g be a multilinear polynomial with 0,1 coefficients. We write $\downarrow g$ for the *son* of g, i.e., the smallest hereditary polynomial containing g. An homogeneous polynomial is a polynomial on which every monomial have the same degree.

Lemma 4. *Let g_n be a sequence of multilinear homogeneous polynomials with* 0,1 *coefficients. Then*

$$(g_n) \leq_c (\downarrow g_n)$$

Proof. Let n be an integer and d the degree of all the monomials of g_n. The generators of $\downarrow g_n$ are simply the monomials of degree d, $g_n = \mathbf{HC}_d(\downarrow g_n)$. The reduction is given by Lemma 2. □

The reduction $(g_n) \leq_c (\downarrow g_n)$ is probably not true in the general case: for instance $f_n = (\prod_{i=1}^n x_i) + per(\bar{x})$ is VNP-complete, but its son is the sum of all monomials of size less than n, and it is therefore in VAC_0. However, we can hope to generalize this inequality to all *pure* polynomials, which we define as polynomials with no pair of comparable monomials.

In our case, we try to demonstrate that $f_n^{\bullet\!-\!\bullet}$ is VNP-complete. This polynomial is hereditary. In Lemma 6, we will demonstrate that the polynomial F_n, which enumerates all graphs composed of some isolated vertices and a complete bipartite connected component, is VNP-complete. Now $f_n^{\bullet\!-\!\bullet}$ is the son of this polynomial: $\downarrow F_n = f_n^{\bullet\!-\!\bullet}$.

Unfortunately this polynomial F_n is not pure, some of its monomials are comparable. In Lemma 8 we demonstrate that the polynomial (G_n), enumerating all the complete bipartite graphs, is also VNP-complete. $f_n^{\bullet\!-\!\bullet}$ is also the son of G_n, but this polynomial is pure!

As we have no general proof of the reduction from a pure polynomial to its son (yet), we will now give an ad hoc demonstration of the reduction from G_n to $f_n^{\bullet\!-\!\bullet}$.

At the end, we will have demonstrated the following chain, for any graph H with no loop and at least one edge.

$$(\mathrm{GF}(K_n, \text{clique})) \leq_c (F_n) \leq_p (\mathrm{Cut}_n^2) \leq_c (G_n) \leq_c (f_n^{\bullet\!-\!\bullet}) \leq_c (f_n^H)$$

As the generating function of clique is VNP-complete, we will have demonstrated Theorem 2.

Lemma 5. *Let $f_n^{\bullet\!-\!\bullet}$ be the polynomial enumerating all graphs of size n homomorphic to $\bullet\!-\!\bullet$. Let G_n be the polynomial enumerating all complete bipartite graphs of size n. Then*

$$(G_n) \leq (f_n^{\bullet\!-\!\bullet})$$

Proof. G_n can be written as $G_n\left((y_e)_{e\in E(K_n)}\right) = \frac{1}{2}\sum_{V\subset[n]}\prod_{v\in V}\prod_{v'\in V^c} y_{v,v'}$. We express G_n in the following way:

$$G_n = \left(\sum_{k=1}^{n-1} \mathbf{HC}^{\bar{x}}_{(n-k)k}\left(\mathbf{HC}^{y}_{k}\left(\mathbf{HC}^{z}_{n-k}\left(\mathbf{HC}^{w}_{1}\left(f_{n+2}^{\bullet\!-\!\bullet}(\bar{x})\right)\right)\right)\right)\right)_{|w,y,z=1}$$

Where $x_{n+1,n+2} = w$ and for all $i \in [n]$, $x_{i,n+1} = y$ and $x_{i,n+2} = z$.

Indeed, $f_n^{\bullet\!-\!\bullet}$ enumerates all bipartite graphs. Each operation above consisting in taking a homogeneous component eliminates some graphs from the list. We will demonstrate that, at the end, the polynomial will only enumerate complete bipartite graphs.

In a first step, we add two vertices $n+1$ and $n+2$ and we give a weight w to the edge between them. By taking the homogeneous component of degree 1 in the variable w, we eliminate all graphs but those which have an edge between $n+1$ and $n+2$.

Then we use variable substitutions to label the edges leaving $n+1$ with the weight y and those leaving $n+2$ with z. A vertex linked to $n+1$ cannot be linked to $n+2$, as our graphs are bipartite and the vertices $n+1$ and $n+2$ are linked. Let us write V and V' for the partition of vertices defined by the bipartite graph, with $n+2 \in V$ and therefore, as $n+1$ and $n+2$ are linked, $n+1 \in V'$.

By taking next the homogeneous component of degree $n-k$ in z, we force that $|V'| \geq n-k$. Similarly, by taking the homogeneous component of degree k in y, we force that $|V| \geq k$. Therefore, as V and V' are disjoint sets, we force that $V^c = V'$.

At the end, by taking the homogeneous component of degree $(n-k)k$ in the variables $\bar{x}$, we only keep bipartite graphs built on V and V^c, with $|V| = k$ and which have $(n-k)k$ edges, i.e., complete bipartite graphs on n vertices.

The reduction is then given by Lemma 2. □

5.2 Bipartite Graphs

Let us demonstrate that F_n is VNP-complete by reducing the generating function of clique to F_n. The proof of the following lemma is omitted because of space restrictions, but it is given in Appendix A.

Lemma 6. *Consider the polynomial*

$$F_n\left((x_e)_{e\in E(K_n)}\right) = GF(K_n, \textit{complete bipartite}) = \sum_{E'} \prod_{e\in E'} x_e,$$

where the summation is done over all subsets $E' \subset E(K_n)$ such that the graph $([n], E')$ (i.e., the graph with n vertices and E' as set of edges) has one complete bipartite connected component and all other connected components reduce to a single vertex.

This p-family is VNP-complete under c-reductions.

As we mentioned before the son of F_n is $f_n^{\bullet\!-\!\bullet}$ (i.e., $\downarrow F_n = f_n^{\bullet\!-\!\bullet}$), but F_n is not pure. We therefore had to introduce the polynomial G_n which is still a generator of $f_n^{\bullet\!-\!\bullet}$ but is pure. For now we introduce another polynomial, the cut enumerator. We will use it to prove the completeness of G_n. But as mentioned in the introduction and in Theorem 1, this family had been introduced by Bürgisser [Bür00] as a family neither VP nor VNP-complete in $\mathbb{F}_2$, if of course VP ≠ VNP.

Be aware that, contrary to the rest of this paper, this polynomial will be built on non-symmetric variables: $x_{i,j}$ will, generally, not be equal to $x_{j,i}$.

Lemma 7. *Let Cut_n^2 be the following polynomial*

$$Cut_n^2\left((x_{i,j})_{i,j\in[n]}\right) = \sum_{V\subset[n]} \prod_{v\in V} \prod_{v'\in V^c} x_{v,v'}$$

This p-family is VNP-complete under c-reductions.

Proof. We will exhibit a p-reduction from $F_n = \mathrm{GF}(K_n, \text{complete bipartite})$ to Cut_{2n}^2. Because $\mathrm{GF}(K_n, \text{complete bipartite})$ is complete under c-reductions, as seen in Lemma 6, this will prove the lemma.

For that, we evaluate Cut_{2n}^2 on $K_{n,n}$ on which we label the edges the following way (remember we are in an oriented graph):

1. A edge between two vertices of the same side of $K_{n,n}$ is labeled by 1,
2. Any edge from the right side to left is also labeled by 1,
3. A horizontal edge from the left side to the right side (i.e., an edge between the i-th left vertex and the i-th right vertex) is labeled by 0,
4. And finally, the edge from the i-th left vertex to the j-th right vertex (with $i \neq j$) is labeled by $y_{i,j}$.

Let $V \subset V(K_{n,n})$. First, if the i-th left vertex is in V and not the i-th right vertex then by (3) $\omega(V) := \prod_{v \in V} \prod_{v' \in V^c} x_{v,v'} = 0$. Therefore, when $\omega(V) \neq 0$, V is of type $V_1 \cup V_2$ with V_1 in the left side, V_2 in the right side and V_1', the image of V_1 on the right side, (i.e., if the i-th left vertex is in V_1, then the i-th right vertex is in V_1') is a subset of V_2. Then if we write V_1^c the complement of V_1 in the left side and V_2^c the one of V_2 in the left side,

$$\begin{aligned}\omega(V) =& \prod_{v \in V} \prod_{v' \in V^c} x_{v,v'} \\ =& (\prod_{i \in V_1} \prod_{v' \in V^c} x_{i,v'})(\prod_{i' \in V_2} \prod_{v' \in V^c} x_{i',v'}) \\ =& (\prod_{i \in V_1} \prod_{j \in V_1^c} x_{i,j})(\prod_{i \in V_1} \prod_{j' \in V_2^c} x_{i,j'})(\prod_{i' \in V_2} \prod_{v' \in V^c} x_{i',v'})\end{aligned}$$

In the last line, the first parenthesis, $(\prod_{i \in V_1} \prod_{j \in V_1^c} x_{i,j})$, takes value 1 because of (1). The second one, $(\prod_{i \in V_1} \prod_{j \in V_2^c} x_{i,j})$, takes value $(\prod_{i \in V_1} \prod_{j \in V_2^c} y_{i,j})$ by (4).The last one, $(\prod_{i \in V_2} \prod_{v' \in V^c} x_{i,v'})$, takes value 1 because of (1) and (2).

Then for every non empty subset V of $V(K_{n,n})$, $\omega(V) = \prod_{i \in V_1} \prod_{j \in V_2^c} y_{i,j}$. Furthermore, there is a bijection between $\{V \subset V(K_{n,n}), \omega(V) \neq 0\}$ and $\{(V, V'), V \subset [n], V' \subset [n], V \cap V' = \emptyset\}$. And this bijection conserves weights. Thus, for the values of the variables $\bar{x}$ defined above,

$$\mathrm{Cut}_{2n}^2(\bar{x}) = \mathrm{GF}(K_n, \text{complete bipartite})(\bar{y}) = F_n$$

□

Lemma 8. *Let G_n be the polynomial enumerating all complete bipartite graphs of size n. Then the p-family (G_n) is VNP-complete under c-reductions.*

The proof is in appendix B.

Perspectives

We have left open in section 5.1 the question of whether a pure multilinear polynomial is always reducible to its son, though we have shown the reduction for homogeneous polynomials and in the special case of homomorphism polynomials. This question can be seen as a generalization of the relationship between the permanent and the partial permanent PER* (cf [Bür00]). Note that the partial permanent is useful to show the completeness of other families of polynomials, so it would be interesting to obtain a general result.

The opposite question, namely whether the son of a polynomial always reduces to it, is also quite interesting. For instance, PER is in VP in characteristic 2. The same result for PER* was implicitly solved in [Val01]. A reduction from $\downarrow f$ to f in the general case would provide another proof of this result and extend our understanding of these polynomials.

Another question raised by this paper is the relation between polynomials which list a graph property as homomorphic polynomials or generating functions on one hand and enumeration on the other. A brief foray into this subject has been made with Corollary 1.

We intend to extend the study of these two questions in our future research.

Acknowledgements. I thank both of my doctoral advisors, A. Durand and G. Malod as well as L. Lyaudet for all his corrections.

References

[Bag09] Bagan, G.: Algorithmes et complexité des problèmes d'énumération pour l'évalution de requêtes logique. PhD thesis, Université de Caen/Basse-Normandie (2009)

[BG04] Bulatov, A.A., Grohe, M.: The Complexity of Partition Functions. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 294–306. Springer, Heidelberg (2004), doi:10.1007/978-3-540-27836-8-27

[BK09] Briquel, I., Koiran, P.: A Dichotomy Theorem for Polynomial Evaluation. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 187–198. Springer, Heidelberg (2009)

[Bri11] Briquel, I.: Complexity issues in counting, polynomial evaluation and zero finding. PhD thesis, Lyon (2011)

[Bür00] Bürgisser, P.: Completeness and Reduction in Algebraic Complexity Theory. Springer (2000)

[CMR01] Courcelle, B., Makowsky, J.A., Rotics, U.: On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. Discrete Applied Mathematics 108(1–2), 23–52 (2001)

[DG00] Dyer, M.E., Greenhill, C.S.: The complexity of counting graph homomorphisms. Random Struct. Algorithms 17(3-4), 260–289 (2000)

[DM11] Durand, A., Mengel, S.: On polynomials defined by acyclic conjunctive queries and weighted counting problems. CoRR, abs/1110.4201 (2011)

[FV98] 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 28(1), 57–104 (1998)

[GT11] Grohe, M., Thurley, M.: Counting homomorphisms and partition functions. CoRR, abs/1104.0185 (2011)

[HN04] Hell, P., Nesetril, J.: Graphs and Homomorphisms. Oxford University Press (2004)

[MRar] Mahajan, M., Raghavendra Rao, B.V.: Small-space analogues of valiant's classes and the limitations of skew formulas. Computational Complexity (to appear)

[Poi08] Poizat, B.: À la recherche de la définition de la complexité d'espace pour le calcul des polynômes à la manière de valiant. Journal of Symbolic Logic 73 (2008)

[Sch78] Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC 1978, pp. 216–226. ACM, New York (1978)

[Val79a] Valiant, L.G.: Completeness classes in algebra. In: Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, STOC 1979, pp. 249–261. ACM, New York (1979)

[Val79b] Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8(2), 189–201 (1979)

[Val01] Valiant, L.G.: Quantum computers that can be simulated classically in polynomial time. In: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, STOC 2001, pp. 114–123. ACM, New York (2001)

[Wel93] Welsh, D.J.A.: Complexity: knots, colourings and counting. London Mathematical Society Lecture Note Series. Cambridge University Press (1993)

A Appendix: Proof of Lemma 6

Lemma (Lemma 6)**.** *Consider the polynomial*

$$F_n\left((x_e)_{e\in E(K_n)}\right) = GF(K_n, \textit{complete bipartite}) = \sum_{E'}\prod_{e\in E'} x_e$$

where the summation is done over all subsets $E' \subset E(K_n)$ such that the graph $([n], E')$ (i.e., the graph with n vertices and E' as set of edges) has one complete bipartite connected component and all other connected components reduce to a single vertex.

This p-family is VNP-complete under c-reductions.

Proof. We will reduce the clique generating function to our polynomial: $g_n(\bar{x}) = \mathrm{GF}(K_n, \mathrm{clique})((x_e)_{e\in E(K_n)}) = \sum_{E'}\prod_{e\in E'} x_e$, where the summation is on all $E' \subset E(K_n)$ such that the graph $([n], E')$ has a complete connected component (a clique) and all others reduce to a single vertex. Bürgisser has demonstrated in [Bür00] that this sequence is VNP-complete under p-reductions.

Let us rewrite our polynomial:

$$F_n(\bar{x}) = \frac{1}{2}\sum_{\substack{V,V'\subset[n]\\ V\cap V'=\emptyset}}\prod_{v\in V}\prod_{v'\in V'} x_{(v,v')}$$

The $\frac{1}{2}$ comes from the fact that each complete bipartite graph can be defined by two different couples of sets: (V, V') and (V', V); and that each couple of vertex sets correspond to a unique complete bipartite graph.

We will proceed to the reduction by beginning with the sequence (F_n) and writing a series of sequences of polynomials $(f_n^k)_{n\in\mathbb{N}}$, each being reducible to the previous one.

Hereafter, we suppose that the left vertices of $K_{n,n}$ are $[1,n]$ and the right ones are $[n+1,2n]$. We write $\widehat{i}$ for the i-th right vertex, i.e., $\widehat{i} = i+n$. Similarly, if $[n]$ are the left vertices of $K_{n,n}$, we write $\widehat{[n]}$ for the right ones, i.e., $\widehat{[n]} = [n+1,2n]$.

$$f_n^1(\bar{x}) = F_{2n}(K_{n,n}) = \sum_{V\subset[n]} \sum_{V'\subset\widehat{[n]}} \prod_{i\in V} \prod_{j\in V'} x_{i,j}$$

Let V and V' be two disjoint sets of vertices. If V has some of its vertices both in the right side and the left side (let us say for instance i in the left side), then we can suppose that V' has a vertex j in the left side. Hence $x_{(i,j)} = 0$ appears in the weight of V', V. Therefore this weight is zero.

The only complete bipartite graphs enumerated in $F_{2n}(K_{n,n})$ are those built on V and V' with V a subset of the left vertices and V' a subset of the right ones. The summation on every subset V of the left vertices avoids repetitions and thus the $\frac{1}{2}$ coefficient is not necessary.

$$f_n^2(\bar{x}, y) = f_{n+1}^1(\bar{x}, x_{n+1,\widehat{i}} = 0, x_{i,\widehat{n+1}} = y) = \sum_{V\subset[n+1]} \sum_{V'\subset\widehat{[n+1]}} \prod_{i\in V} \prod_{j\in V'} x_{i,j}$$

If V and V' are vertex subsets, let $\omega(V, V')$ be their weight: $\omega(V, V') := \prod_{i\in V} \prod_{j\in V'} x_{i,j}$. Let $V \subset [n+1]$.

- If $n+1 \in V$ then $\forall V' \subset \widehat{[n+1]}$, $\omega(V, V') = 0$.
- If $\widehat{n+1} \in V'$ then $\forall V \subset [n]$, $\omega(V, V') = (\prod_{i\in V} \prod_{j\in V'-\{\widehat{n+1}\}} x_{i,j}) \times y^{|V|}$.
- Else, for all $V \subset [n]$ $\deg_y(\omega(V, V')) = 0$.

In those polynomials, the degree of y is a witness to the size of V. We now just have to take the homogeneous component in y of degree k to keep all the V of size k:

$$f_n^{3,k}(\bar{x}) = \mathbf{HC}_k^y\left(f_n^2\right)(\bar{x}, y=1) = \sum_{\substack{V\subset[n]\\|V|=k}} \sum_{V'\subset\widehat{[n]}} \omega(V, V')$$

Let us act similarly for V'.

$$f_n^{4,k}(\bar{x}, y) = f_{n+1}^{3,k}(\bar{x}, x_{n+1,\widehat{i}} = y, x_{i,\widehat{n+1}} = 0)$$

Here the degree of y is a witness to the size of V' in monomials where y is of degree at least 1. Then

$$f_n^{5,k}(\bar{x}) = \mathbf{HC}_k^y\left(f_n^{4,k}\right)(\bar{x}, 1) = \sum_{V\subset[n],|V|=k} \sum_{V'\subset\widehat{[n]},|V'|=k} \omega(V, V')$$

Let us write now

$$f_n^{6,k}(\bar{x}, y) = f_n^{5,k}(\bar{x}, x_{i,\widehat{i}} = y) = \sum_{V\subset[n],|V|=k} \; \sum_{V'\subset\widehat{[n]},|V'|=k} \; \prod_{i\in V} \; \prod_{\substack{j\in V'\\ j\neq i}} x_{i,j} \times y^{|V\cap V'|}$$

And at last, we find the clique generating function:

$$g_n(\bar{x}) = \sum_{k=1}^{n} \mathbf{HC}_k^y \left(f_n^{6,k}\right)(\bar{x}, 1)$$

Indeed, in this polynomial we sum on all subsets $V \subset [n]$ of size k and the degree of y is a witness to the size of $\widehat{V} \cap V'$. Then if we keep only monomials for which the degree in y is k, we only keep complete bipartite graphs built on V and V', with $|V| = |V'| = |\widehat{V} \cap V'|$, i.e., on the same subset of vertices V, on the left and on the right. Then:

$$\sum_{k=1}^{n} \mathbf{HC}_k^y \left(f_n^{6,k}\right)(\bar{x}, 1) = \sum_{V\subset[n]} \prod_{i\in V} \prod_{j\in V} x_{i,\widehat{j}}$$

This polynomial is, by definition, $g_n(\bar{x})$, the clique generating function. Furthermore, as we announce it at the beginning of this demonstration, we have the following reductions:

$$f_n^2 \leq_p f_n^1 \leq_p F_n$$

And, for all n, $L^{f_n^2}(f_n^{3,k}) = \mathcal{O}(nk)$ then $L^{f_{n+1}^2}(f_n^{4,k}) = \mathcal{O}(nk)$. Similarly $L^{f_n^{4,k}}(f_n^{5,k}) = \mathcal{O}(nk)$ then $L^{f_{n+1}^2}(f_n^{6,k}) = \mathcal{O}(n^2k^2)$. At last $L^{f_{n+1}^2}(g_n) = \sum_{k=1}^{n} \mathcal{O}(n^3k^3) = \mathcal{O}(n^7)$.

Thus we have shown a c-reduction from g_n to F_n. □

B Appendix: Proof of Lemma 8

Lemma (Lemma 8)**.** *Let (G_n) be the polynomial enumerating every complete bipartite graphs, i.e.,*

$$G_n((x_{i,j})_{(i,j)\in E(K_n)}) = \frac{1}{2} \sum_{V\subset[n]} \prod_{v\in V} \prod_{v'\in V^c} x_{v,v'}$$

And let Cut_n^2 be the same polynomial but built on oriented graphs, the cut enumerator, i.e.,

$$Cut_n^2((x_{i,j})_{i,j\in[n]}) = \sum_{V\subset[n]} \prod_{v\in V} \prod_{v'\in V^c} x_{v,v'}$$

Then, $(Cut_n^2) \leq_c (G_n)$.

Proof. Here we want to compute the oriented version of G_n knowing how to compute G_n. One of the classical technic is to double each vertex into an input vertex and an output vertex.

First let us create a new graph with $2n$ vertices. In K_{2n}, we say that the n-th first vertices are the left ones and the n-th last vertices are the right ones. The edge between the i-th left vertex and the j-th right is labeled with the oriented variable $x_{i,j}$. Edges between vertices of the same side are labeled with 1 and horizontal edges, i.e., edges between the i-th left vertex and the i-th right are labeled with 0, as we don't want loops. Let K' be this graph to which we add two new vertices, a and b. The new edges are labeled by:

- The edge between a and b is labeled by a new variable t,
- The edge between a and any left vertex by y,
- The edge between a and any right vertex by y',
- The edge between b and any left vertex by z,
- The edge between b and any right vertex by z',

G_n evaluated on this graph will enumerate a list of complete bipartite graphs. By taking homogeneous components we will reduce this list until we find the cut enumerator.

By taking the homogeneous component of degree 1 in t, we only keep in our list the graphs which have a edge between a and b.

For a $k \leq n$, we form a partition of the left side of K' into two sets by joining k vertices of the left side to a and $n-k$ left vertices to b (i.e., by taking the homogeneous component of degree k in y and the homogeneous component of degree $n-k$ in z). A vertex cannot be linked to both a and b, as our graphs must always be bipartite. We name V the set of left vertices linked to a, and therefore V^c is the set of left vertices linked to b.

Similarly, we split the right side of the graph K' into two sets: V', the set of the k right vertices linked to a and V'^c the set of the $n-k$ right vertices linked to b.

A vertex in V cannot be linked to a vertex in V' as they are both linked to a. Similarly a vertex in V^c cannot be linked to a vertex in V'^c because they are both linked to b. Furthermore, every edge between a vertex of V and a vertex of V^c, or between a vertex of V' and a vertex of V'^c is labeled by 1, as they are on the same side. Therefore, the only edges labeled with a $x_{i,j}$ variable that participate in G_n are those between a vertex of V and a vertex of V'^c and those between one of V' and one of V'^c.

If the i-th left vertex of K' is in V and the i-th right vertex of K' is in V'^c, the monomial which is computed by V will have $x_{i,i} = 0$ and therefore is null. Thus if we identify both the left and the right side to $[n]$, $V \cap V'^c = \emptyset$ and then $V = V'$. Finally, if we evaluate G_{2n+2} on K', if we take all the previous homogeneous components and if we evaluate the variables t, y, y', z, and z' to 1, we have

$$\frac{1}{2} \sum_{V \subset [n]} \left(\prod_{i \in V} \prod_{j \in V'^c} x_{i,j} + \prod_{i \in V'} \prod_{j \in V^c} x_{i,j} \right) = \mathrm{Cut}^2(\bar{x})$$

□

Finite State Transducers for Modular Möbius Number Systems

Martin Delacourt[1] and Petr Kůrka[2]

[1] Laboratoire d'Informatique Fondamentale de Marseille, 39 rue Joliot Curie, F-13453 Marseille Cedex, France

[2] Center for Theoretical Study, Academy of Sciences and Charles University in Prague, Jilská 1, CZ-11000 Praha 1, Czechia

Abstract. Modular Möbius number systems consist of Möbius transformations with integer coefficients and unit determinant. We show that in any modular Möbius number system, the computation of a Möbius transformation with integer coefficients can be performed by a finite state transducer and has linear time complexity. As a byproduct we show that every modular Möbius number system has the expansion subshift of finite type.

Keywords: exact real algorithms, expansion subshift, absorptions, emissions.

1 Introduction

In an unpublished but influential manuscript, Gosper [1] shows that continued fractions can be used for arithmetical algorithms, provided they are redundant. Based on these ideas, **exact real arithmetical algorithms** have been developed in Vuillemin [15], Kornerup and Matula [4] or Potts [13]. These algorithms perform a sequence of **input absorptions** and **output emissions** and update their inner state, which may be a (2×2)-matrix in the case of a Möbius transformation or a (2×4)-matrix in the case of binary operations like addition or multiplication.

Using the concepts and methods of symbolic dynamics, exact real arithmetic has been generalized in the theory of **Möbius number systems** (MNS) introduced in Kůrka [6] and developed in Kůrka and Kazda [10]. Möbius number systems represent real numbers by infinite words from a one-sided **expansion subshift**. The letters of the alphabet stand for real orientation-preserving Möbius transformations and the concatenation of letters corresponds to the composition of transformations. In Kůrka [7] we have investigated MNS in which rational numbers have periodic or preperiodic expansions and in Kůrka [9] we have characterized MNS whose expansion subshifts are of finite type or sofic.

The time complexity of the unary exact real algorithm which computes a Möbius transformation depends on the growth of its inner state matrices during the computation. Heckmann [2] analyzes this process in positional number systems and proves the **Law of big numbers** (not to be confused with the Law of

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 323–334, 2012.

large numbers), saying that the norm of the state matrix after n absorptions or emissions is at least of the order $r^{n/2}$ for r-ary positional systems. This implies that the bit size of the state matrices grows at least linearly, and arithmetical operations have quadratic time complexity. In Kůrka [8] we have shown that in a general MNS the growth of the state matrices can be slower and we conjectured that the state matrices can even remain bounded. In the present paper we show that this is the case for modular MNS, i.e., MNS whose transformations have integer coefficients and unit determinant. It follows that the unary algorithm can be realized by a finite state transducer and has linear time complexity. This generalizes the results of Raney [14] and complements the results of Konečný [3], who proves (in a slightly different context), that the only differentiable functions computable by finite state transducers are Möbius transformations.

2 Möbius Transformations

The **extended real line** $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$ can be regarded as a projective space, i.e., the space of one-dimensional subspaces of the two-dimensional vector space. On $\overline{\mathbb{R}}$ we have **homogeneous coordinates** $x = (x_0, x_1) \in \mathbb{R}^2 \setminus \{(0,0)\}$ with equality $x = y$ iff $\det(x,y) = x_0 y_1 - x_1 y_0 = 0$. We regard $x \in \overline{\mathbb{R}}$ as a column vector, and write it usually as $x = \frac{x_0}{x_1} = x_0/x_1$, for example $\infty = 1/0$. The **stereographic projection** $\mathbf{h}(z) = (iz+1)/(z+i)$ maps $\overline{\mathbb{R}}$ to the unit circle $\partial\mathbb{D} = \{z \in \mathbb{C} : \ |z| = 1\}$ in the complex plane, and the upper half-plane $\mathbb{U} = \{z \in \mathbb{C} : \ \Im(z) > 0\}$ conformally to the unit disc $\mathbb{D} = \{z \in \mathbb{C} : \ |z| < 1\}$.

A real **orientation-preserving Möbius transformation** (MT) is a self-map of $\overline{\mathbb{R}}$ of the form

$$M_{(a,b,c,d)}(x) = \frac{ax+b}{cx+d} = \frac{ax_0 + bx_1}{cx_0 + dx_1},$$

where $a, b, c, d \in \mathbb{R}$ and $\det(M_{(a,b,c,d)}) = ad - bc > 0$. Möbius transformations form a group and act also on the upper half-plane $\mathbb{U}$: If $z \in \mathbb{U}$ then $M(z) \in \mathbb{U}$ as well. On $\overline{\mathbb{D}} := \mathbb{D} \cup \partial\mathbb{D}$ we get **disc Möbius transformations** defined by $\widehat{M}_{(a,b,c,d)}(z) = \mathbf{h} \circ M_{(a,b,c,d)} \circ \mathbf{h}^{-1}(z) = (\alpha z + \beta)/(\overline{\beta} z + \overline{\alpha})$, where $\alpha = (a+d) + (b-c)i$, $\beta = (b+c) + (a-d)i$. The **circle derivation** of $M = M_{(a,b,c,d)}$ at $x \in \overline{\mathbb{R}}$ is defined by

$$M^{\bullet}(x) = |\widehat{M}'(\mathbf{h}(x))| = \frac{(ad-bc)\cdot(x_0^2 + x_1^2)}{(ax_0 + bx_1)^2 + (cx_0 + dx_1)^2} = \frac{\det(M)\cdot ||x||^2}{||M(x)||^2}.$$

The **expansion interval** of an MT is $\mathbf{V}(M) = \{x \in \overline{\mathbb{R}} : \ (M^{-1})^{\bullet}(x) > 1\}$. If $M = R_\alpha = M_{(\cos\frac{\alpha}{2}, \sin\frac{\alpha}{2}, -\sin\frac{\alpha}{2}, \cos\frac{\alpha}{2})}$ is a rotation, then $M^{\bullet}(x) = 1$ and $\mathbf{V}(M)$ is empty. Otherwise $\mathbf{V}(M)$ is a proper set interval.

3 Intervals

A **set interval** is an open connected subset of $\overline{\mathbb{R}}$. A **proper set interval** is a nonempty set interval properly included in $\overline{\mathbb{R}}$. We represent proper set intervals by (2×2)-matrices whose columns are their left and right endpoints. The stereographic projection applied to $x = \frac{r\sin\alpha}{r\cos\alpha} \in \overline{\mathbb{R}}$ gives $\mathbf{h}(x) = \sin 2\alpha - i\cos 2\alpha = e^{i(2\alpha - \frac{\pi}{2})}$, so it doubles the angles. Matrices with columns $x = \frac{r\sin\alpha}{r\cos\alpha}$, $y = \frac{s\sin\beta}{s\cos\beta}$ where $0 \le \alpha < 2\pi$, $\alpha < \beta < \alpha + \pi$ therefore represent all proper intervals. Since $\det(x, y) = rs\sin(\alpha - \beta) < 0$, we define matrix intervals as (2×2)-matrices with negative determinant and write them as pairs $I = (\frac{x_0}{x_1}, \frac{y_0}{y_1})$ of their left and right endpoints $\mathfrak{l}(I) = \frac{x_0}{x_1}$, $\mathfrak{r}(I) = \frac{y_0}{y_1}$. The set of **matrix intervals** is therefore

$$\mathbb{I}(\mathbb{R}) = \{(\tfrac{x_0}{x_1}, \tfrac{y_0}{y_1}) \in \mathrm{GL}(\mathbb{R}, 2) :\ x_0y_1 - x_1y_0 < 0\}.$$

We define the **size** and the **length** of an interval (x, y) by

$$\begin{aligned}\mathrm{sz}(x, y) &= \frac{x_0y_0 + x_1y_1}{x_0y_1 - x_1y_0} = \frac{x \cdot y}{\det(x, y)},\\ |(x, y)| &= \frac{1}{2} + \frac{1}{\pi}\arctan \mathrm{sz}(x, y).\end{aligned}$$

For $x = \frac{r\sin\alpha}{r\cos\alpha}$, $y = \frac{s\sin\beta}{s\cos\beta}$ we get $\mathrm{sz}(x, y) = -\cot(\beta - \alpha) = \tan(\beta - \alpha - \frac{\pi}{2})$, so $|(x, y)| = (\beta - \alpha)/\pi$, provided $0 < \beta - \alpha < \pi$. The length $|I| \in (0, 1)$ of I is an increasing function of the size $\mathrm{sz}(I) \in (-\infty, +\infty)$ of I. A matrix interval $I = (x, y)$ defines an open set interval by $z \in I \Leftrightarrow \det(x, z)\cdot\det(z, y) > 0$, and a closed set interval $z \in \overline{I} \Leftrightarrow \det(x, z)\cdot\det(z, y) \ge 0$. If $I = (\frac{r\sin\alpha}{r\cos\alpha}, \frac{s\sin\beta}{s\cos\beta})$, then $z = \frac{t\sin\gamma}{t\cos\gamma} \in I$ iff either $\alpha < \gamma < \beta$ or $\alpha + \pi < \gamma < \beta + \pi$. If I, J are intervals, then $I \subseteq J$ iff $\mathfrak{l}(I) \in \overline{J}$ and $\mathfrak{r}(I) \in \overline{J}$. In this case $\mathrm{sz}(I) \le \mathrm{sz}(J)$. When we transform intervals, we work with the matrix representations of MT rather than with the transformations themselves. Möbius transformations are represented by matrices

$$\mathbb{M}(\mathbb{R}) = \{M_{(a,b,c,d)} \in \mathrm{GL}(\mathbb{R}, 2) :\ ad - bc > 0\}$$

which act on vectors $x \in \mathbb{R}^2$ by $x \mapsto Mx$. Two matrices represent the same MT if one is a nonzero multiple of the other and the matrix multiplication corresponds to the composition of MT. If $M \in \mathbb{M}(\mathbb{R})$ and $I \in \mathbb{I}(\mathbb{R})$, then MI is the interval which represents the M-image of the set interval of I.

4 Rational Intervals

Denote by $\mathbb{Z}$ the set of integers and by $\overline{\mathbb{Q}} = \{x \in \mathbb{Z}^2 \setminus \{\frac{0}{0}\} :\ \gcd(x) = 1\}$ the set of (homogeneous coordinates of) rational numbers which we understand as a subset of $\overline{\mathbb{R}}$. Here $\gcd(x)$ is the greatest common divisor of x_0 and x_1. The norm of a vector $x \in \overline{\mathbb{Q}}$ is $||x|| = \sqrt{x_0^2 + x_1^2}$. Denote by

$$\begin{aligned}\mathbb{M}(\mathbb{Z}) &= \{M \in \mathrm{GL}(\mathbb{Z}, 2) :\ \gcd(M) = 1,\ \det(M) > 0\},\\ \mathbb{I}(\mathbb{Z}) &= \{I \in \mathrm{GL}(\mathbb{Z}, 2) :\ \gcd(I) = 1,\ \det(I) < 0\}.\end{aligned}$$

The norm of a matrix $M_{(a,b,c,d)} \in \mathrm{GL}(\mathbb{Z}, 2)$ is $||M|| = \sqrt{a^2 + b^2 + c^2 + d^2}$. We have $||MN|| \le ||M|| \cdot ||N||$ for $M, N \in \mathbb{M}(\mathbb{Z})$.

Lemma 1. *If $I \in \mathbb{I}(\mathbb{Z})$ is an interval, then*

$$\sqrt{2 \cdot |\det(I) \cdot \mathrm{sz}(I)|} \leq ||I|| \leq 2 \cdot |\det(I)| \cdot \max\{|\mathrm{sz}(I)|, 1\}.$$

Proof. Let $I = (\frac{a}{c}, \frac{b}{d})$. Then $2 \cdot |\det(I) \cdot \mathrm{sz}(I)| = 2|ab + cd| \leq ||I||^2$, and we get the first inequality. To prove the second inequality, we show that in all cases $\max\{|a|, |b|, |c|, |d|\} \leq |\det(I)| \cdot \max\{|\mathrm{sz}(I)|, 1\}$. If $a = 0$ or $d = 0$ then $0 \neq |bc| = |\det(I)|$ and $|\det(I) \cdot \mathrm{sz}(I)|$ is either $|cd|$ or $|ab|$ and the claim is satisfied. If $b = 0$ or $c = 0$ then $0 \neq |ad| = |\det(I)|$ and $|\det(I) \cdot \mathrm{sz}(I)|$ is either $|cd|$ or $|ab|$ and the claim is satisfied. If $\mathrm{sgn}(ab) \cdot \mathrm{sgn}(cd) > 0$ then

$$|a| \cdot |b| + |c| \cdot |d| = |ab + cd| = |\mathrm{sz}(I) \cdot \det(I)|,$$

and the claim is satisfied. If $\mathrm{sgn}(ab) \cdot \mathrm{sgn}(cd) < 0$ then $\mathrm{sgn}(ad) \cdot \mathrm{sgn}(bc) = \mathrm{sgn}(abcd) = \mathrm{sgn}(ab) \cdot \mathrm{sgn}(cd) < 0$ and $|a| \cdot |d| + |b| \cdot |c| = |ad - bc| = |\det(I)|$, so the claim is satisfied. □

Lemma 2. *If $I \in \mathbb{I}(\mathbb{Z})$, $\mathrm{sz}(I) < 0$ and $x \in I \cap \overline{\mathbb{Q}}$, then $||I|| \leq \sqrt{5} \cdot ||x|| \cdot |\det(I)|$ and $|\mathrm{sz}(I)| \leq \frac{5}{2}||x||^2 \cdot |\det(I)|$.*

Proof. Let $x = \frac{p}{q} \in I = (\frac{a}{c}, \frac{b}{d})$, and set $\alpha = -\det(\frac{a}{c}, \frac{p}{q}) = pc - aq$, $\beta = -\det(\frac{p}{q}, \frac{b}{d}) = qb - pd$, so $\mathrm{sgn}(\alpha \cdot \beta) > 0$. Replacing x by $\frac{-p}{-q}$ if necessary, we can assume that $\alpha > 0$ and $\beta > 0$. Since $\mathrm{sz}(I) < 0$ and $\mathrm{sz}(\frac{0}{1}, \frac{1}{0}) = 0$, either $0 \notin I$ or $\infty \notin I$. Assume first $\infty \notin I$, so $cd = -\det(\frac{a}{c}, \frac{1}{0}) \cdot \det(\frac{1}{0}, \frac{b}{d}) \geq 0$. Since $q \neq 0$, $a = (pc - \alpha)/q$, $b = (pd + \beta)/q$, and $-\det(I) = (\alpha d + \beta c)/q = (\alpha|d| + \beta|c|)/|q|$, so $\alpha, \beta, |d|, |c|$ are bounded by $|q| \cdot |\det(I)|$. It follows that $|a|$ and $|b|$ are bounded by $(|p| + 1) \cdot |\det(I)|$, so $||I||^2 \leq 2(q^2 + p^2 + 2|p| + 1) \cdot \det(I)^2$. Similarly if $0 \notin I$, then $ab = -\det(\frac{a}{c}, \frac{0}{1}) \cdot \det(\frac{0}{1}, \frac{b}{d}) \geq 0$. Since $p \neq 0$, $c = (aq + \alpha)/p$, $d = (qb - \beta)/p$, and $-\det(I) = (\alpha b + \beta a)/p = (\alpha|b| + \beta|a|)/|p|$, so $\alpha, \beta, |a|, |b|$ are bounded by $|p| \cdot |\det(I)|$. It follows that $|c|$ and $|d|$ are bounded by $(|q| + 1) \cdot |\det(I)|$, so $||I||^2 \leq 2(p^2 + q^2 + 2|q| + 1) \cdot \det(I)^2$. In both cases $||I||^2 \leq 5 \cdot ||x||^2 \cdot \det(I)^2$. Similarly we show that $|\mathrm{sz}(I)| \leq \frac{5}{2}||x||^2 \cdot |\det(I)|$. □

5 Subshifts

For a finite alphabet $\mathbb{A}$ denote by $\mathbb{A}^* := \bigcup_{m \geq 0} \mathbb{A}^m$ the set of finite words. Denote λ the empty word : $\mathbb{A}^0 = \{\lambda\}$. The length of a word $u = u_0 \dots u_{m-1} \in \mathbb{A}^m$ is $|u| = m$. We denote by $\mathbb{A}^\mathbb{N}$ the Cantor space of infinite words with the metric $d(u, v) = 2^{-k}$, where $k = \min\{i \geq 0 : u_i \neq v_i\}$. We say that $v \in \mathbb{A}^*$ is a subword of $u \in \mathbb{A}^* \cup \mathbb{A}^\mathbb{N}$ and write $v \sqsubseteq u$, if $v = u_{[i,j)} = u_i \dots u_{j-1}$ for some $0 \leq i \leq j \leq |u|$. The cylinder of $u \in \mathbb{A}^n$ is the set $[u] = \{v \in \mathbb{A}^\mathbb{N} : v_{[0,n)} = u\}$. The **shift map** $\sigma : \mathbb{A}^\mathbb{N} \to \mathbb{A}^\mathbb{N}$ is defined by $\sigma(u)_i = u_{i+1}$. A **subshift** is a nonempty set $\Sigma \subseteq \mathbb{A}^\mathbb{N}$ which is closed and σ-invariant, i.e., $\sigma(\Sigma) \subseteq \Sigma$. If $D \subseteq \mathbb{A}^*$ then $\Sigma_D = \{x \in \mathbb{A}^\mathbb{N} : \forall u \sqsubseteq x, u \notin D\}$ is the subshift (provided it is nonempty) with **forbidden words** D. Any subshift can be obtained in this way. A subshift is uniquely determined by its **language** $\mathcal{L}(\Sigma) = \{u \in \mathbb{A}^* : \exists x \in \Sigma, u \sqsubseteq x\}$. Denote by $\mathcal{L}^n(\Sigma) = \mathcal{L}(\Sigma) \cap \mathbb{A}^n$.

A **labelled graph** over an alphabet $\mathbb{A}$ is a structure $\mathcal{G} = (V, E, s, t, \ell)$, where $V = |\mathcal{G}|$ is the set of vertices, E is the set of edges, $s, t : E \to V$ are the source and target maps, and $\ell : E \to \mathbb{A}$ is a labeling function. The subshift of $\mathcal{G}$ consists of all labels of all paths of $\mathcal{G}$. A subshift is **sofic**, if it is the subshift of a finite labelled graph. A subshift Σ is of **finite type** (SFT) of order p, if its forbidden words have length at most p, i.e., if $\Sigma = \Sigma_D$ for some set $D \subset \mathbb{A}^p$. In this case $u \in \mathbb{A}^{\mathbb{N}}$ belongs to Σ iff all subwords of u of length p belong to $\mathcal{L}(\Sigma)$ (see Lind and Marcus [11] or Kůrka [5]).

A **finite state transducer** is a finite state automaton with a read only input tape in an alphabet $\mathbb{A}$ and a write only output tape in an alphabet $\mathbb{B}$. It is given by a finite labelled graph $\mathcal{G}$ with edges $q \xrightarrow{a/b} r$, where $a \in \mathbb{A} \cup \{\lambda\}$ is an input letter and $b \in \mathbb{B} \cup \{\lambda\}$ is an output letter. We say that the transducer is **deterministic** on a subshift $\Sigma \subseteq \mathbb{A}^{\mathbb{N}}$ if for each $q \in V$ and $u \in \Sigma$ there exists a unique $v = F_{\mathcal{G}}(u) \in \mathbb{B}^{\mathbb{N}}$ such that u/v is the label of an infinite path with source q. Such a transducer determines a continuous mapping $F_{\mathcal{G}} : \Sigma \to \mathbb{B}^{\mathbb{N}}$. For any finite state transducer, the computation of $F_{\mathcal{G}}$ has linear time complexity.

6 Möbius Number Systems

A **Möbius iterative system** over an alphabet $\mathbb{A}$ is a map $F : \mathbb{A}^* \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ or a family of orientation-preserving Möbius transformations $(F_u : \overline{\mathbb{R}} \to \overline{\mathbb{R}})_{u \in \mathbb{A}^*}$ satisfying $F_{uv} = F_u \circ F_v$ and $F_\lambda = \mathrm{Id}$. An **open almost-cover** is a system of open intervals $\mathcal{W} = \{W_a : a \in \mathbb{A}\}$ indexed by the alphabet $\mathbb{A}$, such that $\bigcup_{a \in \mathbb{A}} \overline{W_a} = \overline{\mathbb{R}}$. If $W_a \cap W_b = \emptyset$ for $a \neq b$, then we say that $\mathcal{W}$ is an **open partition**. We denote by $\mathcal{E}(\mathcal{W}) = \{\mathbf{l}(W_a), \mathbf{r}(W_a) : a \in \mathbb{A}\}$ the **set of endpoints** of $\mathcal{W}$.

Definition 1. *A Möbius number system over an alphabet $\mathbb{A}$ is a pair $(F, \mathcal{W})$ where $F : \mathbb{A}^* \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ is a Möbius iterative system and $\mathcal{W} = \{W_a : a \in \mathbb{A}\}$ is an almost-cover, such that $W_a \subseteq \mathbf{V}(F_a)$ for each $a \in \mathbb{A}$. The* **interval cylinder** *of $u \in \mathbb{A}^{n+1}$ is $W_u = W_{u_0} \cap F_{u_0} W_{u_1} \cap \cdots \cap F_{u_{[0,n)}} W_{u_n}$. The* **expansion subshift** *$\mathcal{S}_{\mathcal{W}}$ is defined by $\mathcal{S}_{\mathcal{W}} = \{u \in \mathbb{A}^{\mathbb{N}} : \forall k > 0, W_{u_{[0,k)}} \neq \emptyset\}$. We denote by $\mathcal{L}_{\mathcal{W}} = \mathcal{L}(\mathcal{S}_{\mathcal{W}})$ the language of $\mathcal{S}_{\mathcal{W}}$ and by $\mathcal{L}^n_{\mathcal{W}} = \mathcal{L}^n(\mathcal{S}_{\mathcal{W}})$.*

For $uv \in \mathcal{L}_{\mathcal{W}}$ we have $W_{uv} = W_u \cap F_u W_v$. Given a MNS $(F, \mathcal{W})$, we construct nondeterministically the expansion $u \in \mathcal{S}_{\mathcal{W}}$ of $x = x_0 \in \overline{\mathbb{R}}$ as follows: Choose u_0 with $x \in W_{u_0}$, choose u_1 with $x_1 = F^{-1}_{u_0}(x_0) \in W_{u_1}$, choose u_2 with $x_2 = F^{-1}_{u_1}(x_1) \in W_{u_2}$, etc. Then $x \in W_{u_{[0,n)}}$ for each n, so W_u is the set of points which have expansion u.

Theorem 2 (Kůrka and Kazda [10]). *If $(F, \mathcal{W})$ is a MNS over $\mathbb{A}$, then there exists a continuous map $\Phi : \mathcal{S}_{\mathcal{W}} \to \overline{\mathbb{R}}$ such that for each $u \in \mathcal{S}_{\mathcal{W}}$ and $v \in \mathcal{L}_{\mathcal{W}}$,*

$$\lim_{n\to\infty} F_{u_{[0,n)}}(i) = \Phi(u),\ \{\Phi(u)\} = \bigcap_{n \geq 0} \overline{W_{u_{[0,n)}}},\ \Phi([v] \cap \mathcal{S}_{\mathcal{W}}) = \overline{W_v}.$$

Here i is the imaginary unit. In fact we have $\Phi(u) = \lim_{n\to\infty} F_{u_{[0,n)}}(z)$ for each $z \in \mathbb{U}$, and $\mathbf{h}(\Phi(u)) = \lim_{n\to\infty} \widehat{F}_{u_{[0,n)}}(z)$ for each $z \in \mathbb{D}$. If $(F, \mathcal{W})$ is an MNS then $\lim_{n\to\infty} \max\{|W_u| : u \in \mathcal{L}^n_{\mathcal{W}}\} = 0$. This is an immediate consequence of the uniform continuity of $\Phi : \mathcal{S}_{\mathcal{W}} \to \overline{\mathbb{R}}$.

Definition 3. *We say that a MNS $(F, \mathcal{W})$ over $\mathbb{A}$ is an* **integer MNS** *if its transformations have integer entries and its intervals have rational endpoints, i.e., if $F_a \in \mathbb{M}(\mathbb{Z})$ and $W_a \in \mathbb{I}(\mathbb{Z})$ for each $a \in \mathbb{A}$. We say that an integer MNS is* **modular**, *if all its transformations have unit determinant* $\det(F_a) = 1$.

7 Sofic Möbius Number Systems

Definition 4. *Let $(F, \mathcal{W})$ be an MNS over an alphabet $\mathbb{A}$. An open partition $\mathcal{V} = \{V_p : p \in \mathbb{B}\}$ is an* **SFT refinement** *of $\mathcal{W}$, if the following two conditions are satisfied for each $a \in \mathbb{A}$, $p, q \in \mathbb{B}$:*

1. *If $V_p \cap W_a \neq \emptyset$ then $V_p \subseteq W_a$,*
2. *If $V_p \subseteq W_a$ and $V_q \cap F_a^{-1} V_p \neq \emptyset$ then $V_q \subseteq F_a^{-1} V_p$.*

In this case we say that $(F, \mathcal{W}, \mathcal{V})$ is a **sofic Möbius number system**. *The* **base** *graph $\mathcal{G}_{(\mathcal{W},\mathcal{V})}$ of $(F, \mathcal{W}, \mathcal{V})$ is an $\mathbb{A}$-labelled graph whose set of vertices are letters of $\mathbb{B}$ and whose labelled edges are $p \overset{a}{\to} q$ if $F_a V_q \subseteq V_p \subseteq W_a$. Denote by $\mathbb{C} = \{(p, a) \in \mathbb{B} \times \mathbb{A} : V_p \subseteq W_a\}$ and $\mathcal{S}_{(\mathcal{W},\mathcal{V})} \subseteq \mathbb{C}^{\mathbb{N}}$ the SFT of order two with transitions $(p, a) \to (q, b)$ iff $p \overset{a}{\to} q$.*

Theorem 5 (Kůrka [9]). *If $(F, \mathcal{W})$ is an MNS, then $\mathcal{S}_{\mathcal{W}}$ is a sofic subshift iff there exists an SFT refinement $\mathcal{V}$ of $\mathcal{W}$. In this case $\mathcal{S}_{\mathcal{W}}$ is the subshift of the base graph $\mathcal{G}_{(\mathcal{W},\mathcal{V})}$ and we have a factor map $\pi : \mathcal{S}_{(\mathcal{W},\mathcal{V})} \to \mathcal{S}_{\mathcal{W}}$ given by $\pi(p, a) = a$.*

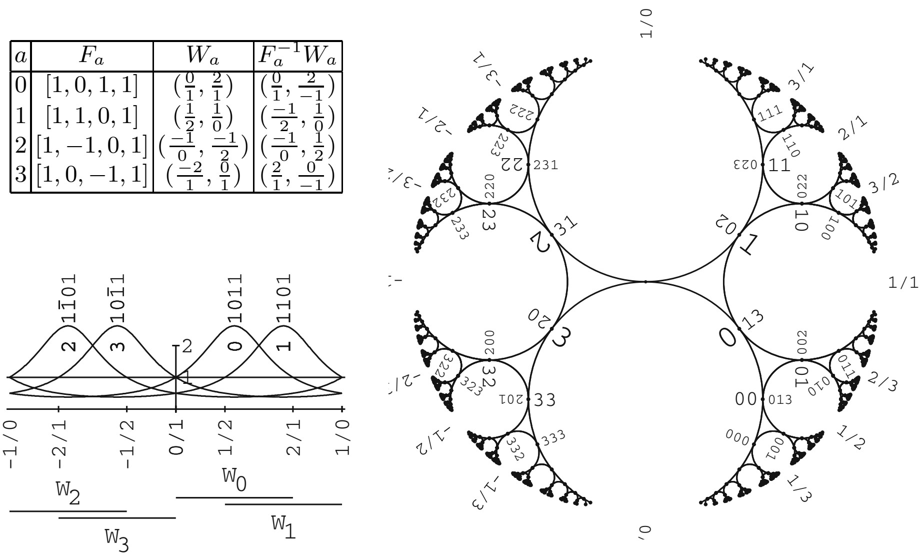
Theorem 6 (Kůrka [9], Theorem 16). *Each modular MNS has a sofic expansion subshift.*

An example of a modular MNS has been studied by Raney [14], Niqui [12] and Kůrka [9]. Its alphabet is $\mathbb{A} = \{0, 1, 2, 3\}$, the transformations are

$$F_0(x) = \frac{x}{1+x},\ F_1(x) = x + 1,\ F_2(x) = x - 1,\ F_3(x) = \frac{x}{1-x}.$$

and the intervals are $W_0 = (0, 1)$, $W_1 = (1, \infty)$, $W_2 = (\infty, -1)$, $W_3 = (-1, 0)$. Since $F_a(0, \infty) = W_a$ for $a = 0, 1$ and $F_a(\infty, 0) = W_a$ for $a = 2, 3$, the expansion subshift is a union of two full subshifts which code respectively nonnegative and nonpositive real numbers: $\mathcal{S}_{\mathcal{W}} = \{0, 1\}^{\mathbb{N}} \cup \{2, 3\}^{\mathbb{N}}$. The system is closely related to continued fractions. Each $u \in \{0, 1\}^{\mathbb{N}}$ can be written as $u = 1^{a_0} 0^{a_1} 1^{a_2} \ldots$, where $a_0 \geq 0$ and $a_n > 0$ for $n > 0$. Then u is the expansion of the continued fraction $[a_0, a_1, a_2, \ldots]$, i.e.,

$$\Phi(u) = [a_0, a_1, a_2, \ldots] = a_0 + 1/(a_1 + 1/(a_2 + \cdots.$$



a	F_a	W_a	$F_a^{-1}W_a$
0	$[1,0,1,1]$	$(\frac{0}{1},\frac{2}{1})$	$(\frac{0}{1},\frac{2}{-1})$
1	$[1,1,0,1]$	$(\frac{1}{2},\frac{1}{0})$	$(\frac{-1}{2},\frac{1}{0})$
2	$[1,-1,0,1]$	$(\frac{-1}{0},\frac{-1}{2})$	$(\frac{-1}{0},\frac{1}{2})$
3	$[1,0,-1,1]$	$(\frac{-2}{1},\frac{0}{1})$	$(\frac{2}{1},\frac{0}{-1})$

Fig. 1. A modular MNS

If $a_n = \infty$ for some $n > 0$, then $\Phi(u) = [a_0, \dots, a_{n-1}]$ is a finite continued fraction.

In Figure 1 we show a variant of this system with larger cylinder intervals $W_a = \mathbf{V}(F_a)$. Figure 1 bottom left shows the graphs of the circle derivations $(F_a^{-1})^\bullet(x)$ together with the cylinder intervals W_a. In Figure 1 right we can see the values $\widehat{F}_u(0)$ of the disc MT $\widehat{F}_u$ at zero. The curves between $\widehat{F}_u(0)$ are constructed as follows. For each MT M there exists a family $(M^r)_{r\in\mathbb{R}}$ of MT such that $M^0 = \mathrm{Id}$, $M^1 = M$, and $M^{r+s} = M^r M^s$. Each value $\widehat{F}_u(0)$ is joined to $\widehat{F}_{ua}(0)$ by the curve $(\widehat{F}_u \widehat{F}_a^r(0))_{0\le r\le 1}$. The labels $u \in \mathbb{A}^*$ at $\widehat{F}_u(0)$ are written in the direction of the tangent vectors $\widehat{F}'_u(0)$. The SFT partition of the system has 8 intervals shown in Figure 2 left. The base graph can be seen in Figure 2 right. The expansion subshift $\mathcal{S}_\mathcal{W}$ is a SFT of order 4. with 20 forbidden words 03, 12, 21, 30, 020, 131, 202, 313, 0220, 0232, 0233, 1322, 1323, 1331, 2002, 2010, 2011, 3100, 3101, 3113.

Theorem 7. *If $(F, \mathcal{W}, \mathcal{V})$ is a modular system, then $\pi : \mathcal{S}_{(\mathcal{W},\mathcal{V})} \to \mathcal{S}_\mathcal{W}$ is an isomorphism, so $\mathcal{S}_\mathcal{W}$ is an SFT.*

Proof. We show that if $(p, u) \in \mathcal{S}_{(\mathcal{W},\mathcal{V})}$, then $p \in \mathbb{B}^\mathbb{N}$ is determined by $u \in \mathbb{A}^\mathbb{N}$. For $0 \le n < m$ we have $V_{p_n} \subseteq W_{u_n}$ and $F_{u_n} V_{p_{n+1}} \subseteq V_{p_n}$, so

$$F_{u_{[n,m)}} V_{p_m} \subseteq F_{u_{[n,m-1)}} V_{p_{m-1}} \subseteq \cdots \subseteq F_{u_n} V_{p_{n+1}} \subseteq V_{p_n},$$
$$F_{u_{[n,m)}} V_{p_m} \subseteq F_{u_{[n,m-1)}} W_{u_{m-1}} \cap \cdots \cap F_{u_n} W_{u_{n+1}} \cap W_{u_n} \subseteq W_{u_{[n,m)}}.$$

It follows that $\emptyset \neq F_{u_{[n,m)}} V_{p_m} \subseteq V_{p_n} \cap W_{u_{[n,m)}}$ is nonempty. Denote by $x_n = \Phi(\sigma^n(u))$, so $\{x_n\} = \bigcap_{m>n} \overline{W_{u_{[n,m)}}}$. If x_n is irrational, then there exists $m > n$

pa	V_p	F_a	$F_a^{-1}V_p$	followers
00	$(\frac{0}{1},\frac{1}{2})$	$[1,0,1,1]$	$(\frac{0}{1},\frac{1}{1})$	0, 1
10	$(\frac{1}{2},\frac{1}{1})$	$[1,0,1,1]$	$(\frac{1}{1},\frac{1}{0})$	2, 3
11	$(\frac{1}{2},\frac{1}{1})$	$[1,1,0,1]$	$(\frac{-1}{2},\frac{0}{1})$	7
20	$(\frac{1}{1},\frac{2}{1})$	$[1,0,1,1]$	$(\frac{1}{0},\frac{2}{-1})$	4
21	$(\frac{1}{1},\frac{2}{1})$	$[1,1,0,1]$	$(\frac{0}{1},\frac{1}{1})$	0, 1
31	$(\frac{2}{1},\frac{1}{0})$	$[1,1,0,1]$	$(\frac{1}{1},\frac{1}{0})$	2, 3
42	$(\frac{-1}{0},\frac{-2}{1})$	$[1,-1,0,1]$	$(\frac{-1}{0},\frac{-1}{1})$	4, 5
52	$(\frac{-2}{1},\frac{-1}{1})$	$[1,-1,0,1]$	$(\frac{-1}{1},\frac{0}{1})$	6, 7
53	$(\frac{-2}{1},\frac{-1}{1})$	$[1,0,-1,1]$	$(\frac{-2}{-1},\frac{-1}{0})$	3
62	$(\frac{-1}{1},\frac{-1}{2})$	$[1,-1,0,1]$	$(\frac{0}{1},\frac{1}{2})$	0
63	$(\frac{-1}{1},\frac{-1}{2})$	$[1,0,-1,1]$	$(\frac{-1}{0},\frac{-1}{1})$	4, 5
73	$(\frac{-1}{2},\frac{0}{1})$	$[1,0,-1,1]$	$(\frac{-1}{1},\frac{0}{1})$	6, 7

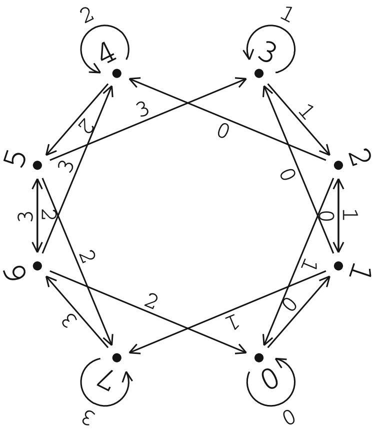


Fig. 2. The SFT partition and the base graph of a modular system from Figure 1

such that $W_{u_{[n,m)}} \cap \mathcal{E}(\mathcal{V}) = \emptyset$, so there exists exactly one $p_n \in \mathbb{B}$ with $V_{p_n} \cap W_{u_{[n,m)}} \neq \emptyset$. Assume that x_n is rational. For each $m > n$ we have

$$x_m = \Phi(\sigma^m(u)) = F^{-1}_{u_{[n,m)}}(x_n) \in \overline{W_{u_m}} \subseteq \overline{\mathbf{V}(F_{u_m})},$$

and $||x_m||^2/||x_{m+1}||^2 = ||x_m||^2/||F^{-1}_{u_m}(x_m)||^2 = (F^{-1}_{u_m})^\bullet(x_m) \geq 1$, so $||x_{m+1}|| \leq ||x_m||$. Moreover, if $x_m \in W_{u_m}$, then $||x_{m+1}|| < ||x_m||$. Since $||x_m||^2 \in \mathbb{N}$, the set $\{m \geq n : \ x_m \in W_{u_m}\}$ is finite and there exists $m > n$ such that either $x_k = \mathfrak{l}(W_{u_k})$ for all $k \geq m$, or $x_k = \mathbf{r}(W_{u_k})$ for all $k \geq m$. Since $x_n = F_{u_{[n,k)}}(x_k) \in \overline{W_{u_{[n,k)}}} \subseteq F_{u_{[n,k)}}\overline{W_{u_k}}$, we get $x_n = \mathfrak{l}(W_{u_{[n,k)}})$ for all $k \geq m$ in the former case and $x_n = \mathbf{r}(W_{u_{[n,k)}})$ for all $k \geq m$ in the latter case. It follows that there exists $k > m$ such that $W_{u_{[n,k)}} \cap \mathcal{E}(\mathcal{V}) = \emptyset$, so there exists a unique p_n with $V_{p_n} \cap W_{u_{[n,k)}} \neq \emptyset$. This means that p_n is uniquely determined by u. Since $F_{u_{n-1}} V_{p_n} \subseteq V_{p_{n-1}}$, the letter p_{n-1} is uniquely determined by p_n and the prefix $p_{[0,n)}$ of p is uniquely determined by p_n. □

Theorem 8. *Assume that $(F, \mathcal{W}, \mathcal{V})$ is a modular MNS and for $u \in \mathcal{L}_\mathcal{W}$ denote by $\mathcal{P}(u) \subseteq \mathbb{B}^*$ the set of paths with label u.*

1. There exists $r > 0$ such that the set $\{p_{[0,n-r]} : \ p \in \mathcal{P}(u)\}$ is a singleton for each $n > r$ and each finite word $u \in \mathcal{L}^n_\mathcal{W}$.
2. There exists $s > 0$ such that $\mathcal{P}(u)$ has at most s elements for each $u \in \mathcal{L}_\mathcal{W}$.
3. The map $\pi^{-1} : \mathcal{S}_\mathcal{W} \to \mathcal{S}_{(\mathcal{W},\mathcal{V})}$ can be computed by a finite state transducer.

Proof. The existence of constants r, s follows from Theorem 7 by a compactness argument. We define a finite state transducer for π^{-1} as follows. Its vertices are sets $X \subseteq \mathbb{B}^n$, where $0 < n \leq r$. The labelled edges are

$$X \overset{a/\lambda}{\longrightarrow} \{p \in \mathbb{B}^{n+1} : \ p_{[0,n-1]} \in X, \ p_{n-1} \overset{a}{\to} p_n\} \quad \text{if } X \subseteq \mathbb{B}^n, \ n < r,$$

$$X \overset{a/b}{\longrightarrow} \{p \in \mathbb{B}^r : \ bp_{[0,r-2]} \in X, \ p_{r-2} \overset{a}{\to} p_{r-1}\} \quad \text{if } X \subseteq \mathbb{B}^r.$$

Then u/p is the label of a path with the source $\mathbb{B}$ iff p is a prefix of a path whose label is u. □

In Table 2 left we show the computation of $\pi^{-1}(u)$ on input word $u = 00133$. For each $n > 0$ we give the set $\mathcal{P}(u_{[0,n)})$ of all paths $p \in \mathbb{B}^{n+1}$ with label $u_{[0,n)}$.

8 Arithmetical Algorithms

Definition 9. *The* **unary graph** *for an integer sofic MNS* $(F, \mathcal{W}, \mathcal{V})$ *is a labelled graph whose vertices are* (X, p)*, where* $X \in \mathbb{M}(\mathbb{Z})$ *and* $p \in \mathbb{B}$*. Its labelled edges are*

$$\begin{aligned} \textit{absorption:}\ & (X,p) \xrightarrow{a/\lambda} (XF_a, q) \quad \textit{if}\ F_aV_q \subseteq V_p \subseteq W_a, \\ \textit{emission:}\ & (X,p) \xrightarrow{\lambda/b} (F_b^{-1}X, p) \ \ \textit{if}\ XV_p \subseteq W_b. \end{aligned}$$

The labels of paths are concatenations of the labels of their edges. They have the form u/v where $u \in \mathcal{L}_{\mathcal{W}}$ is an input word and $v \in \mathcal{L}_{\mathcal{W}}$ is an output word.

Proposition 10. *If* $(X,p) \xrightarrow{u/v} (Y,q)$ *is a path in the unary graph, then*

$$Y = F_v^{-1}XF_u,\ F_uV_q \subseteq V_p \cap W_u,\ XF_uV_q \subseteq W_v.$$

Proof. Since $W_\lambda = \overline{\mathbb{R}}$ and $F_\lambda = \mathrm{Id}$, the statement holds for the absorption and emission edges. Assume by induction that the statement holds for a path with label u/v. If $(X,p) \xrightarrow{u/v} (Y,q) \xrightarrow{a/\lambda} (Z,r)$ then $Z = YF_a = F_v^{-1}XF_{ua}$, $F_aV_r \subseteq V_q \subseteq W_a$, so $F_{ua}V_r \subseteq F_uV_q \subseteq V_p \cap W_u \cap F_uW_a = V_p \cap W_{ua}$, and $XF_{ua}V_r \subseteq XF_uV_q \subseteq W_v$, so the statement holds for $(X,p) \xrightarrow{ua/v} (Z,r)$. If $(X,p) \xrightarrow{u/v} (Y,q) \xrightarrow{\lambda/b} (Z,q)$ then $Z = F_b^{-1}Y = F_{vb}^{-1}XF_u$. From $F_v^{-1}XF_uV_q = YV_q \subseteq W_b$ we get $XF_uV_q \subseteq F_vW_b$, and therefore $XF_uV_q \subseteq W_v \cap F_vW_b = W_{vb}$. Moreover, $F_uV_q \subseteq V_p \cap W_u$, so the statement holds for $(X,p) \xrightarrow{u/vb} (Z,q)$. □

Table 1. The unary algorithm

```
procedure unary;
input: $M \in \mathbb{M}(\mathbb{Z})$, $(p,u) \in \mathcal{S}_{(\mathcal{W},\mathcal{V})} \cup \mathcal{L}_{(\mathcal{W},\mathcal{V})}$;
output: $v \in \mathcal{S}_{\mathcal{W}} \cup \mathcal{L}_{\mathcal{W}}$;
variables $X \in \mathbb{M}(\mathbb{Z})$ (state), $n, m \in \mathbb{N}$ (input and output pointers);
begin
  $X := M$; $n := 0$; $m := 0$;
  while $n < |u|$ repeat
    if $\forall a \in \mathbb{A}, XV_{p_n} \not\subseteq W_a$ then begin
      $X := XF_{u_n}$; $n := n + 1$; end;
    else begin
      $v_m := a$, where $XV_{p_n} \subseteq W_a$ and $XV_{p_n} \not\subseteq W_b$ for all $b < a$;
      $X := F_a^{-1}X$; $m := m + 1$; end;
    end;
```

Table 2. The computation of a path $p = 0017 = \pi^{-1}(001333) = \pi^{-1}(u)$ (left) and the computation of $v = 010222 = \Theta_M(p, u)$ on the input matrix $M(x) = (2x+1)/(x+2)$ and the input path $0 \xrightarrow{0} 0 \xrightarrow{0} 1 \xrightarrow{1} 7 \xrightarrow{3} 7$ by the unary algorithm (right). The third column gives the values v_m on emission steps and the empty word on absorption steps. The last column gives the vertex p_n on emission steps and the edge $p_n \xrightarrow{u_n} p_{n+1}$ on absorption steps.

n	$\mathcal{P}(001333_{[0,n)})$
1	$00, 01, 12, 13, 24,$
2	$000, 001, 012, 013, 124,$
3	$0017, 0120, 0121,$ $0132, 0133,$
4	$00176, 00177,$
5	$001764, 001765,$ $001776, 001777,$
6	$0017653, 0017764,$ $0017765, 0017776,$ $0017777,$

n	m	out	X	XV_{p_n}	input
0	0	0	$[2, 1, 1, 2]$	$(\frac{1}{2}, \frac{4}{5})$	0
0	1	1	$[2, 1, -1, 1]$	$(\frac{1}{1}, \frac{4}{1})$	0
0	2		$[3, 0, -1, 1]$	$(\frac{0}{1}, \frac{3}{1})$	$0 \xrightarrow{0} 0$
1	2	0	$[3, 0, 0, 1]$	$(\frac{0}{1}, \frac{3}{2})$	0
1	3		$[3, 0, -3, 1]$	$(\frac{0}{1}, \frac{3}{-1})$	$0 \xrightarrow{0} 1$
2	3	2	$[3, 0, -2, 1]$	$(\frac{3}{0}, \frac{3}{-1})$	1
2	4	2	$[1, 1, -2, 1]$	$(\frac{3}{0}, \frac{2}{-1})$	1
2	5	2	$[-1, 2, -2, 1]$	$(\frac{3}{0}, \frac{1}{-1})$	1
2	6		$[-3, 3, -2, 1]$	$(\frac{3}{0}, \frac{0}{-1})$	$1 \xrightarrow{1} 7$
3	6		$[-3, 0, -2, -1]$	$(\frac{3}{0}, \frac{0}{-1})$	$7 \xrightarrow{3} 7$

We consider a deterministic **unary algorithm** given in Table 1, which computes a path in the unary graph. Its input is a matrix $M \in \mathbb{M}(\mathbb{Z})$ and either a finite path $(p, u) \in \mathcal{L}_{(\mathcal{W},\mathcal{V})}$ or an infinite path $(p, u) \in \mathcal{S}_{(\mathcal{W},\mathcal{V})}$. We assume that the alphabet $\mathbb{A}$ is linearly ordered. At each step, the algorithm performs the first possible emission if there is one, and an absorption if there is no emission applicable. For an infinite input path, the algorithm computes an output word $v \in \mathcal{S}_{\mathcal{W}}$ such that u/v is the label of a path in the unary graph with source (M, p_0). An example of the computation of the unary algorithm is given in Table 2 right. We are going to prove that for a modular system $(F, \mathcal{W}, \mathcal{V})$, the norm of the state matrix X remains bounded during the computation of the unary algorithm. To do so, we define some constants and prove several lemmas. Set

$$\begin{aligned} B_0 &= \max\{\sqrt{5} \cdot ||x|| : \ x \in \mathcal{E}(\mathcal{W})\}, & B_1 &= \max\{1, |\mathrm{sz}(F_b^{-1} W_b)| : \ b \in \mathbb{A}\} \\ D_0 &= \min\{|\det(V_p)| : \ p \in \mathbb{B}\}, & D_1 &= \max\{|\det(V_p)| : \ p \in \mathbb{B}\}, \\ G &= \max\{1, ||V_p^{-1} F_a V_q|| : \ p \xrightarrow{a} q\}, & H &= \max\{\sqrt{D_0}, ||V_p|| : \ p \in \mathbb{B}\}, \\ B &= \max\{B_0, 2B_1\}, & C_0 &= \max\{B^2 D_1^2 G^2 / 2D_0, B_1\} \end{aligned}$$

Lemma 3. *1. If $(X, p) \xrightarrow{a/\lambda} (XF_a, q)$, then $\mathrm{sz}(XF_aV_q) < \mathrm{sz}(XV_p)$.*
2. If $(X, p) \xrightarrow{\lambda/b} (F_b^{-1}X, p)$, then $0 > \mathrm{sz}(XV_p) < \mathrm{sz}(F_b^{-1}XV_p) < B_1$.

Proof. The first claim follows from $XF_aV_q \subseteq XV_p$. To prove the second claim, note that for each $M \in \mathbb{M}(\mathbb{Z})$ we have $\mathrm{sz}(\mathbf{V}(M)) < 0$, so $\mathrm{sz}(W_b) < 0$ for each $b \in \mathbb{A}$. If $(X, p) \xrightarrow{\lambda/b} (F_b^{-1}X, p)$ is an emission edge, then $XV_p \subseteq W_b$, so $\mathrm{sz}(XV_p) < 0$. Since $F_b^{-1}XV_p \subseteq F_b^{-1}W_b$, we get $\mathrm{sz}(F_b^{-1}XV_p) < B_1$. Since F_b^{-1} is an expansion on W_b, we get $\mathrm{sz}(XV_p) < \mathrm{sz}(F_b^{-1}XV_p)$. □

Lemma 4. *If* $(X,p) \xrightarrow{a/\lambda} (XF_a,q)$ *is an absorption performed by the unary algorithm and* $\mathrm{sz}(XV_p) < B_1$*, then* $||XV_p|| < BD_1 \det(X)$*,* $|\mathrm{sz}(XV_p)| < C_0 \det(X)$ *and* $|\mathrm{sz}(XF_aV_q)| < C_0 \det(X)$*.*

Proof. We distinguish two cases. If $0 \le \mathrm{sz}(XV_p) < B_1$, then by Lemma 1 we have $||XV_p|| < 2|\det(XV_p)| \cdot \max\{1, |\mathrm{sz}(XV_p)|\} \le 2B_1D_1 \det(X)$. If $\mathrm{sz}(XV_p) < 0$, then we use the fact that XV_p is not contained in any W_a, so it must contain a point from $\mathcal{E}(\mathcal{W})$. By Lemma 2, $||XV_p|| \le B_0 \cdot |\det(XV_p)| \le B_0D_1 \det(X)$. Thus in both cases we have $||XV_p|| \le BD_1 \det(X)$. It follows $||XF_aV_q|| \le ||XV_p|| \cdot ||V_p^{-1}F_aV_q|| \le BD_1G \cdot \det(X)$. By Lemma 1 we get $|\mathrm{sz}(XV_p)| \le ||XV_p||^2/2|\det(XV_p)| \le \frac{B^2D_1^2}{2D_0} \det(X) \le C_0 \det(X)$, and similarly $|\mathrm{sz}(XF_aV_q)| \le \frac{B^2D_1^2G^2}{2D_0} \det(X) \le C_0 \det(X)$. □

Lemma 5. *Every infinite path computed by the unary algorithm contains an infinite number of emissions.*

Proof. Assume by contradiction that there exists an infinite path of absorptions with vertices (X_n, p_n) and label u/λ, where $u \in \mathcal{S}_{\mathcal{W}}$. Since $F_{u_{[0,n)}}V_{p_n} \subseteq W_{u_{[0,n)}}$ and $\lim_{n\to\infty} |W_{u_{[0,n)}}| = 0$, we get $\lim_{n\to\infty} |X_0F_{u_{[0,n)}}V_{p_n}| = 0$ by the continuity of X_0, and therefore $\lim_{n\to\infty} \mathrm{sz}(X_0F_{u_{[0,n)}}V_{p_n}) = -\infty$. This is in a contradiction with Lemma 4. □

Theorem 11. *For a modular MNS* $(F, \mathcal{W}, \mathcal{V})$ *there exists a constant* $C > 0$ *such that for every input matrix* $M \in \mathbb{M}(\mathbb{Z})$*, the unary algorithm computes a continuous function* $\Theta_M : \mathcal{S}_{(\mathcal{W},\mathcal{V})} \to \mathcal{S}_{\mathcal{W}}$ *with* $\Phi\Theta_M(p,u) = M\Phi(u)$*, and the state matrix* X *satisfies* $||X|| < C \cdot \max\{||M||^2, \det(M)^2\}$ *during the computation.*

Proof. Let (X_n, p_n) be the vertices of the infinite path with source $(X_0, p_0) = (M, p_0)$. If $\mathrm{sz}(X_nV_{p_n}) > C_0 \det(M)$, then (X_n, p_n) is an absorption vertex by Lemma 3 and $\mathrm{sz}(X_{n+1}V_{p_{n+1}}) < \mathrm{sz}(X_nV_{p_n})$. If $\mathrm{sz}(X_nV_{p_n}) < -C_0 \det(M)$, then (X_n, p_n) is an emission vertex by Lemma 4, and $\mathrm{sz}(X_{n+1}V_{p_{n+1}}) > \mathrm{sz}(X_nV_{p_n})$. Thus there exists m, such that for all $n \ge m$ we have $|\mathrm{sz}(X_nV_{p_n})| < C_0 \det(M)$ while for $n < m$ we have $|\mathrm{sz}(X_nV_{p_n})| \le |\mathrm{sz}(MV_{p_0})| \le \frac{H^2 \cdot ||M||^2}{2D_0 \det(M)}$. By Lemma 1 we get either $||X_nV_{p_n}|| \le 2D_1C_0 \det(M)^2$ in the former case and $||X_nV_{p_n}|| \le \frac{H^2D_1}{D_0}||M||^2$ in the latter case. Taking $C = \max\{2HD_1C_0, H^3D_1/D_0\}$ we get

$$||X_n|| \le ||X_nV_{p_n}|| \cdot ||V_{p_n}^{-1}|| \le C \cdot \max\{||M||^2, \det(M)^2\}$$

for all n, so the algorithm can be realized by a finite state transducer. By Lemma 5, for each $(p,u) \in \mathcal{S}_{(\mathcal{W},\mathcal{V})}$ there exists a unique $v = \Theta_M(p,u)$ such that u/v is the label of an infinite path with source (M, p_0). For each m there exists n such that $u_{[0,n)}/v_{[0,m)}$ is the label of a finite path with source (M, p_0), $\emptyset \ne F_{u_{[0,n)}}V_{p_n} \subseteq W_{u_{[0,n)}}$, and $\emptyset \ne MF_{u_{[0,n)}}V_{p_n} \subseteq W_{v_{[0,m)}}$. The intersection $\bigcap_n F_{u_{[0,n)}}\overline{V_{p_n}} \subseteq \bigcap_n \overline{W_{u_{[0,n)}}}$ is nonempty by compactness and has zero diameter, so it contains the unique point $\Phi(u)$. The intersection $\bigcap_n MF_{u_{[0,n)}}\overline{V_{p_n}} \subseteq \bigcap_m \overline{W_{v_{[0,m)}}}$ is a nonempty singleton which contains both $M(\Phi(u))$ and $\Phi(v)$, so $M(\Phi(u)) = \Phi(v)$. □

Corollary 12. *If* $(F, \mathcal{W}, \mathcal{V})$ *is a modular MNS, then for each* $M \in \mathbb{M}(\mathbb{Z})$ *there exists a finite state transducer which computes a continuous function* $\Psi_M : \mathcal{S}_{\mathcal{W}} \to \mathcal{S}_{\mathcal{W}}$ *which satisfies* $\Phi\Psi_M = M\Phi$*.*

Proof. Using Theorems 8 and 11 we get $\Psi_M = \Theta_M \circ \pi^{-1}$.

A disadvantage of modular systems is that they are not redundant. As shown in Kůrka [7], the cylinder intervals of a modular system contain neither 0 nor ∞, so they cannot form a cover but only an almost-cover. In Kůrka [8] we argue that in some redundant MNS, the unary algorithm has asymtotically linear time complexity. The norm of the state matrix remains small most of the time, although fluctuations to larger values occur sporadically.

References

1. Gosper, R.W.: Continued fractions arithmetic. Unpublished manuscript (1977), http://www.tweedledum.com/rwg/cfup.htm
2. Heckmann, R.: Big integers and complexity issues in exact real arithmetic. Electr. Notes Theor. Comput. Sci. 13 (1998)
3. Konečný, M.: Real functions incrementally computable by finite automata. Theoretical Computer Science 315(1), 109–133 (2004)
4. Kornerup, P., Matula, D.W.: An algorithm for redundant binary bit-pipelined rational arithmetic. IEEE Transactions on Computers 39(8), 1106–1115 (1990)
5. Kůrka, P.: Topological and symbolic dynamics, Cours spécialisés, vol. 11. Société Mathématique de France, Paris (2003)
6. Kůrka, P.: Möbius number systems with sofic subshifts. Nonlinearity 22, 437–456 (2009)
7. Kůrka, P.: Expansion of rational numbers in Möbius number systems. In: Kolyada, S., Manin, Y., Moller, M. (eds.) Dynamical Numbers: Interplay between Dynamical Systems and Number Theory. Contemporary Mathematics, vol. 532, pp. 67–82. American Mathematical Society (2010)
8. Kůrka, P.: Fast arithmetical algorithms in Möbius number systems. IEEE Transactions on Computers 61(8), 1097–1109 (2012)
9. Kůrka, P.: Stern-Brocot graph in Möbius number systems. Nonlinearity 25, 57–72 (2012)
10. Kůrka, P., Kazda, A.: Möbius number systems based on interval covers. Nonlinearity 23, 1031–1046 (2010)
11. Lind, D., Marcus, B.: An Introduction to Symbolic Dynamics and Coding. Cambridge University Press, Cambridge (1995)
12. Niqui, M.: Exact real arithmetic on the Stern-Brocot tree. J. Discrete Algorithms 5(2), 356–379 (2007)
13. Potts, P.J.: Exact real arithmetic using Möbius transformations. PhD thesis, University of London, Imperial College, London (1998)
14. Raney, G.N.: On continued fractions and finite automata. Mathematische Annalen 206, 265–283 (1973)
15. Vuillemin, J.E.: Exact real computer arithmetic with continued fractions. IEEE Transactions on Computers 39(8), 1087–1105 (1990)

Zero-Knowledge Proofs via Polynomial Representations

Giovanni Di Crescenzo[1] and Vadym Fedyukovych[2]

[1] Applied Communication Sciences, New Jersey, USA
gdicrescenzo@appcomsci.com
[2] GlobalLogic, Kiev, Ukraine
vf@unity.net

Abstract. Under the existence of commitment schemes with homomorphic properties, we construct a constant-round zero-knowledge proof system for an $\mathcal{NP}$-complete language that requires a number of commitments that is *sublinear* in the size of the (best known) witness verification predicate. The overall communication complexity improves upon best known results for the specific $\mathcal{NP}$-complete language [1,2] and results that could be obtained using zero-knowledge proof systems for the entire $\mathcal{NP}$ class (most notably, [3,2,4]). Perhaps of independent interest, our techniques build a *proof system* after reducing the theorem to be proved to statements among low-degree polynomials over large fields and using Schwartz-Zippel lemma to prove polynomial identities among committed values.

1 Introduction

The fascinating notion of zero-knowledge proofs has been introduced in the seminal paper [5]. A zero-knowledge proof is a method allowing a prover to convince a polynomial time bounded verifier that a certain statement $x \in L$ is true without revealing any additional information (such as all or any part of the witness, when L is in $\mathcal{NP}$). The wide applicability of this notion was realized thanks to another seminal paper [1], where it was proved that the $\mathcal{NP}$-complete language of 3-colorable graphs, and thus all languages in $\mathcal{NP}$, have a zero-knowledge proof, under the existence of commitment schemes, which, in turn, exist under certain reasonable complexity-theoretic assumptions. This result opened the door to a general and powerful methodology for protecting cryptographic protocols against malicious adversaries, by using zero-knowledge proofs for $\mathcal{NP}$ statements related to the specific protocol. Since then, cryptographic applications have demanded the design of improved zero-knowledge protocols in terms of one or more of a number of metrics; most notably, round complexity (starting with, e.g., [6,7]), time complexity (starting with, e.g., [8]), and communication complexity (starting with, e.g., [3,2]). In this paper we consider the problem of obtaining communication-efficient zero-knowledge proof systems for languages in $\mathcal{NP}$, while keeping both round and time complexity reasonably efficient. In this area, [3] first produced a proof system that requires a number of commitments that is subquadratic in the circuit size m, [2] produced a proof system that requires a number of commitments only slightly super-linear in m, and [4] achieved a number of commitments linear in m, which might seem the best possible performance, given that each circuit gate has to contribute to the protocol and contributing through a constant number of commitments

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 335–347, 2012.

per gate seems the minimum required from any zero-knowledge guarantee. However, perhaps surprisingly, [9] recently improved the overall communication complexity of [4] to $O(m+\text{poly}(\ell)\text{polylog}(k))$, where ℓ is the security parameter, k is the soundness parameter, and 'poly' denotes a potentially large polynomial. In general, m is a polynomial in the witness size n, and so is also ℓ, as one needs the probability that the zero-knowledge property does not hold to be negligible in n. Thus, the result in [9], as also mentioned by the authors, only improves the result from [4] whenever m is a large polynomial in n. This is unfortunate as for many natural problems in $\mathcal{NP}$, the circuit size m is a relatively small polynomial in n (e.g., for 3SAT m is linear in n, and for many graph-based $\mathcal{NP}$-complete problems m is only quadratic in n, as scanning the graph adjacency matrix suffices to verify the instance's witness). The natural question left open from [4,9] remains whether there exists a natural problem in $\mathcal{NP}$ which has a zero-knowledge proof requiring a number of commitments that is sublinear in the circuit size m. In this paper, we solve this problem for the $\mathcal{NP}$-complete language of q-colorability.

Our Contribution and Related Work. Assuming the existence of commitment schemes with certain homomorphic properties, we design a constant-round zero-knowledge proof system for the $\mathcal{NP}$-complete language of q-colorable graphs with communication complexity about $O((\ell+k)(nq+N))$ (see Theorem 1 for exact quantities), where n and N are the number of nodes and edges of the input graph, ℓ is the length of each commitment, and k is such that the soundness holds with probability $\leq 2^{-k}$. The number of required commitments is equal to $O(nq+N)$, which is smaller, for all $3 \leq q \leq N/n$, than the size of the smallest circuit for verifying a q-coloring of the input graph, which is, to the best of our knowledge, $O(N \log q)$. For instance, when $q = N^{\delta}$, for some $0 < \delta < 1$, our scheme needs $O(N)$ commitments while the circuit size is $O(N \log N)$.

Our protocol improves the communication complexity of two protocols that can be obtained by extending previous proof systems for the related 3-colorability language. Specifically, it improves over the natural extension of the two previous 3-colorability protocols [1,2] by a factor of $O(kn)$ and $O(k \log q)$, respectively. (See also top part of Figure 1 for a more detailed comparison.) It also improves over the natural extension of results from previous papers [3,2,4,9] that construct a proof system for an arbitrary language L in $\mathcal{NP}$ by working directly on the circuit that verifies the witness for the input statement. (See bottom part of Figure 1 for more detailed comparisons.) Here, our protocol improves the performance in [4] by a factor of $\log q$, for all values of m, and thus even when [9] does not improve over [4]. Moreover, our protocol improves the performance in [9] for the case of q-colorability, as the large polynomial in the poly(ℓ) factor in the communication complexity of the protocol in [9] is asymptotically larger than m and is thus larger than our protocol's performance. (On the other hand, for large values of m, we do not improve the performance of this protocol which remains the best known.) Finally, we remark that, through reducibility, our protocol directly applies to all languages with reduce to q-colorability, via a suitably economical reduction.

The honest-verifier zero-knowledge property of our protocol holds under the existence of commitment schemes with certain properties, that holds, for instance, under intractability of the Decisional Diffie-Hellman problem; and the zero-knowledge

Paper:	[1]	[2]	This paper
Communication complexity:	$O(nNk\ell)$	$O((n+Nk)\ell(\log q))$	$O(\ell(nq+N))$

Paper	[4]	[9]	This paper
Communication complexity	$O(m\ell)$	$O(m+\text{poly}(\ell)\text{polylog}(k))$	$O(\ell(m/\log q))$

Fig. 1. The top table compares our result for the language of q-colorable graphs with natural extensions of previous proof systems for the language of 3-colorable graphs. Here, n is the number of nodes, N the number of edges of the input graph, k is the parameter such that soundness holds with probability $\leq 2^{-k}$ and ℓ is the length of commitments. The bottom table compares our result for the language of q-colorable graphs with natural extensions of previous proof systems for an arbitrary language in $\mathcal{NP}$. Here, m is the size of the best circuit for q-col, which is, to the best of our knowledge, $O(N\log q)$.

property can be obtained under the same assumption using techniques from [7]. Our proof system is based on a non-trivial combination of 3 techniques, as we now explain.

The first technique is the reduction of the graph q-colorability statement to a statement among certain low-degree polynomials over large fields. Reducing a language membership statement to a statement about a single low-degree polynomial over large fields has been a successful technique in the area of interactive proof systems (that are not necessarily zero-knowledge): there, two famous results showed that all languages in the polynomial hierarchy [10] and all languages in PSPACE [11] have an interactive proof system. We could not use the same technique as it does not preserve zero-knowledge. Among the substantial differences, here we employ two different polynomials: one for the prover and one for the verifier, precisely to preserve zero-knowledge properties. A related technique was first designed in [12,13] to provide *arguments* (i.e., proofs only sound against polynomial-time algorithms) for the languages of graph isomorphism and hamiltonian graphs. In this paper we design a variant of this technique to provide a *proof* for the language of graph q-colorability, and first specialize it to show that this technique can provide savings in communication complexity.

The second technique is the use of commitment schemes with certain hiding, binding, homomorphic and provability properties. It turns out that the commitment schemes from [14] suffice for our purposes even though we have to uncover and use certain additional homomorphic and provability properties. Related commitment schemes, requiring some of the same homomorphic and provability properties, but dual hiding and binding properties, were used in [15]. Other related and earlier protocols, also using homomorphic properties of discrete logarithms, include [8,16,17].

The third technique in our protocol is the use of Schwartz-Zippel lemma [18] to analyze the proofs of polynomial identities among committed values over large fields, and is essentially yet another variation over the numerous previous applications of the Schwartz-Zippel lemma in cryptographic or computer science research papers.

2 Definitions and Preliminaries

In this section we recall known definitions and results that are of interest for the description of our main result. After giving some basic definitions, we recall the definitions of zero-knowledge proof systems and commitment schemes.

Graph Colorability. Let Γ be a graph, and let $\mathbb{V}(\Gamma)$ and $\mathbb{E}(\Gamma)$ denote the (ordered) set of vertices and the (ordered) set of edges of graph Γ, with size n and N, respectively. For any q (possibly a function of n), we say that a function $\phi : \mathbb{V}(\Gamma) \rightarrow \{1, \ldots, q\}$ is a *q-coloring* of graph Γ if for each $(u, v) \in \mathbb{E}(\Gamma)$, it holds that $\phi(u) \neq \phi(v)$. Deciding whether an input graph has a q-coloring, for $q \geq 3$, is a known family of $\mathcal{NP}$-complete problems. Computing the chromatic number (i.e., the smallest q such that the graph has a q-coloring) is also known to be $\mathcal{NP}$-complete. A language L is a subset of $\{0, 1\}^*$. We denote by q-COL the languages of graphs Γ that have a q-coloring.

Computational Indistinguishability. We say that a function f is *negligible* in n if for any constant c there exists a constant n_0 such that $f(n) \leq n^{-c}$, for all positive integers $n \geq n_0$. We say that two (families of) distributions $D_0 = \{D_{0,n}\}_{n\in\mathbb{N}}, D_1 = \{D_{1,n}\}_{n\in\mathbb{N}}$ are *computationally indistinguishable* if for all efficient non-uniform (distinguishing) algorithms A, the difference below is negligible:

$$\left|\text{Prob}\left[\, z \leftarrow D_{0,n} : A(z) = 1 \,\right] - \text{Prob}\left[\, z \leftarrow D_{1,n} : A(z) = 1 \,\right]\right|.$$

Zero-Knowledge Proofs. We use the notions of *interactive Turing machine* and *interactive protocol* given in [5]. If A and B are two interactive probabilistic Turing machines, by pair (A,B) we denote an interactive protocol. Let x be an input common to A and B. Informally, an interactive proof system [5] for a language L is an interactive protocol in which a prover convinces a polynomial-time bounded verifier that a common input string x belongs to L. An interactive proof system has to satisfy the two requirements of *completeness* and *soundness*. Completeness says that if $x \in L$ then the verifier returns ACCEPT at the end of the protocol with high probability (i.e., at least $1 - \epsilon_c$, for some small completeness error ϵ_c). Soundness says that if $x \notin L$ then for any (computationally unlimited) prover strategy, the verifier returns ACCEPT at the end of the protocol with small probability (i.e., at most ϵ_s, for some small soundness error ϵ_s). Given an interactive proof system with constant completeness and soundness errors (e.g., $\epsilon_c = 1/3$ and $\epsilon_s = 1/2$), one can apply $O(k)$ independent sequential or parallel repetitions, and majority calculations, to transform it into another proof system with exponentially small errors $\epsilon_c = \epsilon_s = 2^{-k}$.

A zero-knowledge proof system [5] for a language L is an interactive proof system for L in which, informally speaking, no polynomial-time verifier obtains any additional information other than the fact that $x \in L$. This is formalized in the zero-knowledge requirement, which says that for all input $x \in L$, and any polynomial-time verifier, this verifier's view during the protocol can be simulated by a polynomial-time algorithm, called the simulator. Specifically, the distribution consisting of the output of the simulator on input x is computationally indistinguishable from the distribution consisting of this verifier's random coins and the messages exchanged by the prover and this verifier using common input x. That is, the probability that an efficient adversary can tell apart these two distributions is negligible in the security parameter ℓ, which is linear (or polynomial) in the input length. Zero-knowledge proof systems are called *honest-verifier* zero-knowledge, if the zero-knowledge property holds only with respect to the honest verifier V, as opposed to all polynomial-time verifiers V′. Interactive proof systems are called *public-coin* if the verifier's program only consists of sending his random coins.

The public-coin, honest-verifier, zero-knowledge proof system for 3-COL from [1] achieves communication complexity $O((n + Nk)\ell)$, and can be naturally extended to language q-COL with communication complexity $O((n + Nk)\ell(\log q))$.

Commitment Schemes. Informally speaking, a bit commitment scheme (A,B) is a two-phase interactive protocol between two probabilistic polynomial time parties A and B, called the committer and the receiver, respectively, such that the following is true. In the first phase (the commitment phase), A commits to a string s by computing a pair of keys (com, dec) and sending com (the commitment key) to B. In the second phase (the decommitment phase) A reveals the string s and the key dec (the decommitment key) to B. Now B checks whether the decommitment key is valid; if not, B outputs a special string $\perp$, meaning that he rejects the decommitment from A; otherwise, B can efficiently compute the string s revealed by A. (A,B) has to satisfy three requirements: *correctness*, saying that A can successfully decommit the committed value, *(computational) hiding*, saying that B's views of the commitment phase run on two different committed strings are computationally indistinguishable, and *statistical binding*, saying that no algorithm A can decommit to two different values in the decommitment phase, unless with negligible probability. Many variants of this definition of commitment schemes have been studied in the cryptography literature; here, we consider a variant where the generation of the commitment and decommitment keys may be done in two messages (one from the receiver, then one from the committer) and the revealing of the decommitment key is done in a single message from the committer to the receiver.

3 Main Techniques

We describe the main ingredients in our construction for q-COL: a commitment scheme with special homomorphic and provability properties, using low-degree polynomials to prove and verify the 3-colorability statement about the input graph, and using Schwartz-Zippel lemma to prove/verify that a polynomial evaluates to zero.

3.1 A Commitment Scheme

We use a computationally-hiding and perfectly-binding commitment scheme that further satisfies certain homomorphic and provability properties. Specifically, we will additionally require that the committed value belongs to a field $\mathbb{Z}_Q$, and that atomic commitments can be homomorphically composed into commitments to low-degree polynomials over values in $\mathbb{Z}_Q$, which is formally expressed as follows:

0) given degree d, value $z \in \mathbb{Z}_Q$, and commitments to coefficients $a_0, \ldots, a_d \in \mathbb{Z}_Q$, it is possible to efficiently compute a commitment to $\sum_{i=0}^{d} a_i z^i \in \mathbb{Z}_Q$.

We note that a commitment scheme enjoys the above property if it enjoys the following two homomorphism properties:

1) given a commitment to a value $a \in \mathbb{Z}_Q$ and a commitment to a value $b \in \mathbb{Z}_Q$, it is possible to efficiently compute a commitment to value $a + b \mod Q$;
2) given a commitment to a value $a \in \mathbb{Z}_Q$ and a value $c \in \mathbb{Z}_Q$, it is possible to efficiently compute a commitment to value $a \cdot c \mod Q$.

Furthermore, we need the commitment scheme to enjoy various provability properties with respect to the type of statement to be proved and the type of protocol used to prove this statement. With respect to the type of statement, we would need to prove that a commitment com can be opened as a given value $a \in \mathbb{Z}_Q$ or to prove certain polynomial equations about committed values in $\mathbb{Z}_Q$. With respect to the type of protocol used, we would need a communication-efficient, 3-message, public-coin, honest-verifier zero-knowledge proof system, with a structure similar to the protocol in [8].

Commitment Construction. We use the commitment construction from [14], based on discrete logarithms, and, more specifically on a variant of El-Gamal encryption and Pedersen's commitment [17]. This scheme can be described as follows.

On input a (unary) security parameter ℓ, algorithm A uniformly chooses ℓ-bit primes P, Q such that $P = 2Q+1$, and generators g, h of the Q-order subgroup of Z_P. Then, to commit to a value $v \in \{0, \ldots, Q-1\}$, A uniformly chooses $r \in \{0, \ldots, Q-1\}$, computes $\hat{g} = g^r \mod P$ and $\hat{h} = h^{r+v} \mod P$, and outputs: $com = (P, Q, g, h, \hat{g}, \hat{h})$ and $dec = (r, v)$. When many commitments are computed, algorithm A can continue using the same 4-tuple (P, Q, g, h) and generate a new pair $(\hat{g}, \hat{h})$ for each value to be committed. (In the rest of the paper, we always consider groups Z_P, Z_Q for commitment and committed values and thus omit the $\mod P$ and $\mod Q$ suffixes, respectively.)

Now we show that this commitment scheme satisfies the above homomorphic properties (1) and (2). Let (P, Q, g, h) be the 4-tuple shared in all commitments. First, given a commitment $com_a = (\hat{g}_a, \hat{h}_a)$ to a value $a \in Z_Q$ and a commitment $com_b = (\hat{g}_b, \hat{h}_b)$ to a value $b \in Z_Q$, it is possible to efficiently compute a commitment $com_{a+b} = (\hat{g}_{a+b}, \hat{h}_{a+b})$ to value $a + b$, by setting $\hat{g}_{a+b} = \hat{g}_a \hat{g}_b$ and $\hat{h}_{a+b} = \hat{h}_a \hat{h}_b$. Second, given a commitment $com_a = (\hat{g}_a, \hat{h}_a)$ to a value $a \in Z_Q$ and a value $c \in Z_Q$, it is possible to efficiently compute a commitment $com_{ac} = (\hat{g}_{ac}, \hat{h}_{ac})$ to value $a \cdot c$, by setting $\hat{g}_{ac} = \hat{g}_a^c$ and $\hat{h}_{ac} = \hat{h}_a^c$.

We will also need a honest-verifier zero-knowledge proof to prove that a commitment key can be decommitted as a certain value. Specifically, to open a commitment $(\hat{g}, \hat{h})$ as value v, the prover uses r as an auxiliary input in the following 3-round protocol (essentially identical to a protocol in [14]):

1. The prover uniformly chooses $s \in \mathbb{Z}_Q$, computes $(\bar{g}, \bar{h}) = (g^s, h^s)$ and sends $(\bar{g}, \bar{h})$ to the verifier.
2. The verifier uniformly chooses a challenge $q \in \mathbb{Z}_Q$ and sends it to the prover.
3. The prover responds by sending $a = qr + s$.
4. The verifier accepts if $\hat{g}^q \bar{g} = g^a$ and $\hat{h}^q \bar{h} = h^a h^{qv}$.

In [14], two facts are shown about this protocol: (1) it has optimal soundness, in that if a commitment key cannot be opened as v, then a dishonest prover can make the verifier accept with probability at most $1/|\mathbb{Z}_Q|$; (2) it is honest-verifier zero-knowledge, in that an efficient simulator who knows the verifier's coins can generate a triple distributed identically to the triple $((\bar{g}, \bar{h}), q, a)$ in the protocol. Moreover, we need a proof system with responses consisting of linear polynomials in the challenge, to prove/verify statements about committed colours. We consider the following protocol:

1. Prover uniformly chooses $\alpha, \beta \in \mathbb{Z}_Q$ and computes first message $(\bar{g}, \bar{h}) = (g^\beta, h^{\beta+\alpha})$. Prover sends $(\bar{g}, \bar{h})$ to Verifier.
2. Verifier chooses and sends a challenge $q \in \mathbb{Z}_Q$.
3. Prover produces and sends responses $b = qr + \beta$, $a = qv + \alpha$.
4. Verifier accepts if $\hat{g}^q\bar{g} = g^b$ and $\hat{h}^q\bar{h} = h^{b+a}$.

Lemma 1. *The prover's response value a is a polynomial that is linear in the challenge q, the top coefficient being the committed value v.*

Proof. As the group is cyclic, one can always write $\hat{g} = g^e, \bar{g} = g^f, \hat{h} = h^c, \bar{h} = h^d$ for some c, d, e, f. It follows that $b - (eq + f) = 0$ and $(b + a) - (cq + d) = 0$. Then we obtain that $a = q(c - e) + (d - f)$. Thus, the lemma follows. □

3.2 Graph Chromatic Polynomials

Let c_v be a colour assigned to a vertex $v \in \mathbb{V}(\Gamma)$ and let α_v be an initial random value of the proof system. We consider colours and initial random values to be elements of a finite field. We assign a linear univariate polynomial over the field that depends on colours of vertices connected to each edge $a = (H_a, T_a)$ of the graph. We then assign a product of such polynomials to the whole graph.

Definition 1. *For any graph Γ, colours c_v and initial random values α_v assigned to vertices v of the graph, we define the* graph chromatic polynomial *of Γ as*

$$f_C(z, \Gamma) = \prod_{(H_a,T_a)\in\mathbb{E}(\Gamma)} ((zc_{H_a} + \alpha_{H_a}) - (zc_{T_a} + \alpha_{T_a}))$$

It is clear that the degree of the graph chromatic polynomial is at most number of edges in graph Γ.

Lemma 2. *For any graph Γ, and for any colours and initial random values assigned to vertices in $\mathbb{V}(\Gamma)$, the degree of the graph chromatic polynomial is equal to the number $N = |\mathbb{E}(\Gamma)|$ if and only if the assigned colours constitute a proper graph coloring.*

Proof. It is straightforward to check that top coefficient of the graph chromatic polynomial is $\prod_{(H_a,T_a)\in\mathbb{E}(\Gamma)}(c_{H_a} - c_{T_a})$. This coefficient is zero if and only if there is an edge in the graph with the same colours assigned to adjacent vertices. □

For any graph having a proper coloring, let u be the inverse of the top coefficient of the graph chromatic polynomial and let ψ be the related initial random value.

Definition 2. *For any graph Γ, colours c_v and initial random values α_v assigned to vertices v of the graph, inverse u and related initial random value ψ, we define the* graph chromatic verification polynomial *of Γ as*

$$f_V(z,\Gamma) = (zu+\psi) \prod_{(H_a,T_a)\in\mathbb{E}(\Gamma)} ((zc_{H_a}+\alpha_{H_a}) - (zc_{T_a}+\alpha_{T_a}))$$

It is clear that for any graph with proper coloring, (1) the degree of graph chromatic verification polynomial is exactly $N+1$, and (2) the top coefficient of the graph chromatic verification polynomial is exactly 1.

Lemma 3. *For any non properly colored graph Γ, any u, any ψ and any $\{m_j\}_{j=0\ldots N}$, it holds that*

$$f_V(z,\Gamma) \not\equiv z^{N+1} + \sum_{j=0}^{N} m_j z^j \qquad (1)$$

We re-state our initial problem of deciding graph q-colorability as an equivalent problem of deciding whether the degree of the graph chromatic polynomial equals the number of its edges. We do that by testing graph chromatic verification polynomial degree to be $N+1$ and top coefficient to be 1. We observe that the prover's responses available to the verifier while running a protocol for the commitment scheme in Section 3.1 are polynomials that are linear in the verifier's challenge (i.e., $C_v(z) = zc_v + \alpha_v$ for any $v \in \mathbb{V}(\Gamma)$). A key point here is that the verifier can evaluate the graph chromatic verification polynomial only using prover's responses, without access to prover's witness (i.e., the coloring). To decide this problem, the verifier would evaluate chromatic verification polynomial at some point chosen by him as a challenge $z = z_0$.

Let $B = \{b_1,\ldots,b_q\}$ be a set of pre-defined colours that are a part of proper coloring of a graph. To each vertex in Γ, we assign a polynomial that depends on its color.

Definition 3. *Let Γ be a graph. For any vertex $v \in \mathbb{V}(\Gamma)$, we define the* vertex chromatic polynomial *of vertex v of graph Γ as*

$$f_B(z,v,\Gamma) = \prod_{b_l\in B} ((zc_v+\alpha_v) - zb_l).$$

Lemma 4. *For any initial value α_v, the degree of the vertex chromatic polynomial is at most $|B|-1$ (that is, $q-1$) if and only if the colour assigned to the vertex is one of the pre-defined colours.*

Proof. It is straightforward to check that the top coefficient of vertex chromatic polynomial is $\prod_{b_l\in B}(c_v - b_l)$. This coefficient is zero if and only if the colour of the vertex is one of the proper colours: $c_v \in B$. □

We re-state our initial problem of testing whether a colour of a vertex is one of proper colours with an equivalent problem testing whether we have zero coefficient at z^q. We let the prover submit arbitrary commitments to the rest of coefficients, for example, in case of 3-colorability, to "square", "linear" and constant coefficients of vertex chromatic polynomials. Now the verifier can evaluate all such polynomials at a point chosen as a challenge $z = z_0$ to test whether the colour of each vertex is one of the proper colours.

3.3 Low-Degree Polynomial Identity Testing

We test whether a univariate polynomial is identically zero by evaluating it at a random point chosen as a challenge. It is well known that for any non-zero polynomial over a finite field, the degree of the polynomial is an upper bound on the number of roots. It follows, using Schwartz-Zippel lemma [18], that the probability that a uniformly chosen value is a root is at most the polynomial's degree divided by cardinality of the set that challenges are chosen from. In our case, the set chosen is $\mathbb{Z}_Q$ for a large prime Q (exponential in the security parameter), and the polynomial's degree is small (polynomial in the security parameter). Thus, the probability that a uniformly chosen value is a root is negligible in the security parameter. Then we replace deciding original problem of q-colorability with deciding whether a polynomial is identically zero, where different polynomials are used, based on the techniques in Section 3.2. Here, operations over these polynomials are performed in the exponents of the commitment keys, using the homomorphic properties of the commitment scheme from Section 3.1.

4 A Proof System for q-Colorable Graphs

In this section we present our protocol for q-COL, based on the techniques discussed in Section 3. We obtain the following

Theorem 1. *Assuming the hardness of the Decisional Diffie-Hellman problem modulo ℓ-length primes, there exists (constructively) a zero-knowledge proof system for q-COL with soundness error 2^{-k} and communication complexity $O(\max(k+\log n, \ell)\cdot(nq+N))$, where $n = |\mathbb{V}(\Gamma)|$, $N = |\mathbb{E}(\Gamma)|$, and Γ is the common input graph.*

Considering that according to natural setting of parameters k, ℓ, we have that $k+\log n = O(\ell)$, and comparing with the natural extension of the protocol in [2], this protocol improves the communication complexity by a factor of $O(k \log q)$, for any $3 \leq q \leq N/n$. We further note that the intractability assumption in Theorem 1 can be generalized to the existence of commitment schemes with homomorphic and provability properties, as described in Section 2.

To prove Theorem 1, we focus our description on a 3-message, public-coin, honest-verifier zero-knowledge proof system for the same language. This is because such a protocol can be made zero-knowledge with respect to any verifier without increase in the asymptotic communication complexity by applying the technique from [7]. Specifically, the verifier commits to its random coins at the start of the protocol and opens the appropriate commitments whenever the verifier in the honest-verifier protocol would directly send the coins. We first give an informal discussion, then continue with the formal description of our proof system, and then give a discussion of its communication complexity, completeness, soundness and honest-verifier zero-knowledge properties.

An Informal Discussion. Our honest-verifier zero-knowledge proof system can be described as composed of the following three subprotocols. First, the prover commits to all vertices' colours using the commitment scheme from Section 3.1, and proves that he can open the commitment keys using the associated proofs described in Section 3.1. Moreover, the prover shows that the committed colour assigned to each vertex is one of the pre-defined q colours. This is proved using Lemma 4, and thus proving that the degree of vertex chromatic polynomial induced by the committed colours is at most $q-1$ (for example, for 3-colorability, quadratic in the challenge). Finally, the prover shows that the committed colours assigned to all vertices define a valid q-coloring of the graph. This is proved using Lemma 2, and thus proving that the top coefficient of the graph chromatic polynomial is non-zero, which can be checked by the verifier using the graph chromatic verification polynomial. A critical part of our approach is being able to show that all edges have different colors by showing a single (and much shorter) proof that the top coefficient of the graph chromatic polynomial is non-zero; instead, a generic circuit incurs a multiplicative factor of $\log q$ to test this difference for all edges. We also note that our polynomial representation approach does *not* imply a smaller circuit for verifying that a graph is q-colorable: the underlying circuit is, in fact, larger as it consists of computing a chromatic polynomial (which takes time $O(N(\log q)^2)$) and testing that the top coefficient is nonzero (which takes $O(\log q)$), for a total of $O(N(\log q)^2)$.

Formal Description. The input common to prover and verifier is a graph Γ. The prover's auxiliary input is a q-coloring ϕ of Γ.

1. The prover defines a permutation η from $\{1,\dots,q\}$ to a set $B=\{b_1,\dots,b_q\}\subset\mathbb{Z}_Q$ of colours and generates the commitment scheme parameters (P,Q,g,h) as in Section 3.1. For each vertex $v\in\mathbb{V}(\Gamma)$, the prover defines $c_v=\eta(\phi(v))$, uniformly chooses $r_v,\alpha_v,\beta_v\in\mathbb{Z}_Q$ and commits to colour c_v and value α_v by computing

$$(\hat{g}_v,\hat{h}_v)=(g^{r_v},h^{r_v+c_v}),\qquad(\tilde{g}_v,\tilde{h}_v)=(g^{\beta_v},h^{\beta_v+\alpha_v}).\tag{2}$$

The prover computes coefficients $\{w_{vk}\}_{v\in\mathbb{V}(\Gamma),k=0\dots q-1}$ of vertex chromatic polynomials:

$$\prod_{b_l\in B}((zc_v+\alpha_v)-zb_l)=\sum_{k=0}^{q-1}z^k w_{vk}\tag{3}$$

The prover uniformly chooses $\{\delta_{vk}\}_{v\in\mathbb{V}(\Gamma),k=0\dots q-1}$, and computes commitments $\{(\zeta_{vk},\gamma_{vk})\}$:

$$\zeta_{vk}=g^{\delta_{vk}},\qquad\gamma_{vk}=h^{\delta_{vk}+w_{vk}}.\tag{4}$$

The prover computes the inverse of the top coefficient of the graph chromatic polynomial:

$$u=\left(\prod_{(H_a,T_a)\in\mathbb{E}(\Gamma)}(c_{H_a}-c_{T_a})\right)^{-1}\tag{5}$$

The prover uniformly chooses $\nu,\psi,\mu\in\mathbb{Z}_Q$ and commits to the inverse of the top coefficient:

$$(\hat{\lambda},\hat{\sigma})=(g^{\nu},h^{\nu+u}),\qquad(\tilde{\lambda},\tilde{\sigma})=(g^{\mu},h^{\mu+\psi}).\tag{6}$$

The prover computes coefficients $\{m_j\}_{i=j\ldots N}$ of the graph chromatic verification polynomial as follows:

$$(zu+\psi)\prod_{(H_a,T_a)\in\mathbb{E}(\varGamma)}((zc_{H_a}+\alpha_{H_a})-(zc_{T_a}+\alpha_{T_a}))$$
$$=z^{N+1}+\sum_{j=0}^{N}z^j m_j \quad (7)$$

For $j=0,\ldots,N$, the prover uniformly chooses $\rho_j\in\mathbb{Z}_Q$ and commits to the coefficient m_j:

$$\xi_j=g^{\rho_j},\qquad \pi_j=h^{\rho_j+m_j}. \quad (8)$$

The prover sends $\{(\hat{g}_v,\hat{h}_v,\tilde{g}_v,\tilde{h}_v)\}$, $\{(\zeta_{vk},\gamma_{vk})\}$, $(\hat{\lambda},\hat{\sigma},\tilde{\lambda},\tilde{\sigma})$, $\{(\xi_j,\pi_j)\}$ to the verifier.

2. The verifier uniformly chooses a challenge $z_0\in\mathbb{Z}_Q$ and sends it to the prover.
3. The prover computes and sends responses C_v,R_v,U_v for each $v\in\mathbb{V}(\varGamma)$, and T,V,W, as below:

$$C_v=z_0c_v+\alpha_v,\qquad R_v=z_0r_v+\beta_v,\qquad U_v=\sum_{k=0}^{q-1}z_0^k\delta_{vk}, \quad (9)$$

$$T=\sum_{j=0}^{N}z_0^j\rho_j,\qquad V=z_0\nu+\mu,\qquad W=z_0u+\psi. \quad (10)$$

4. The verifier computes F_v for each $v\in\mathbb{V}(\varGamma)$, and S, as below:

$$F_v=\prod_{b_l\in B}(C_v-z_0b_l),\quad S=W\prod_{(H_a,T_a)\in\mathbb{E}(\varGamma)}(C_{H_a}-C_{T_a}). \quad (11)$$

The verifier accepts if for each $v\in\mathbb{V}(\varGamma)$, it holds that

$$\hat{g}_v^{z_0}\tilde{g}_v=g^{R_v},\qquad \hat{h}^{z_0}\tilde{h}_v=h^{R_v+C_v} \quad (12)$$

$$g^{U_v}\prod_{k=0}^{q-1}\zeta_{vk}^{-z_0^k}=1,\qquad h^{U_v+F_v}\prod_{k=0}^{q-1}\gamma_{vk}^{-z_0^k}=1 \quad (13)$$

$$\hat{\lambda}^{z_0}\tilde{\lambda}=g^V,\qquad \hat{\sigma}^{z_0}\tilde{\sigma}=h^{V+W} \quad (14)$$

$$g^T\prod_{j=0}^{N}\xi_j^{-z_0^j}=1,\qquad h^{T+S-z_0^{N+1}}\prod_{j=0}^{N}\pi_j^{-z_0^j}=1 \quad (15)$$

We note that under the natural parameter setting $k+\log n=O(\ell)$, the communication complexity of the above protocol is $O(\ell)$ in the verifier's message, $O(\ell(N+nq))$ in the first prover's message, and $O(\ell n)$ in the last prover's message. Overall, the number of commitments can be anywhere between $O(nq)$ and $O(n^2)$, depending on the number of edges N, and is always smaller than the size $O(N \log q)$ of the best known verifying circuit by a $\log q$ factor.

In our proof of the soundness property, we count the number of verifier's challenges to which any prover can answer, and use this number to derive an upper bound for the soundness error. First, we note that for any vertex v and for any $(\hat{g}_v, \hat{h}_v, \tilde{g}_v, \tilde{h}_v)$, the acceptable response C_v is always a polynomial that is linear in challenge z_0, as was shown in Lemma 1. For any top coefficient of this polynomial, that is not the colour committed, there is exactly one possible challenge that admits a response C_v acceptable at (12). For n equations at (12), we have at least $Q - n$ 'interesting' challenges such that all responses acceptable at (12) are 'good', that is, are polynomials linear in challenge with top coefficients being the colours committed. Considering a good response acceptable at (13) for a non properly colored vertex, we further reduce interesting challenges set cardinality by at most qn; passing (14) and (15) would reduce by at most $N + 2$. This gives soundness error $((q + 1)n + N + 2)/Q$.

In our proof of the honest-verifier zero-knowledge property, we construct a honest-verifier simulator by producing simulated commitments to 'constant' coefficients of chromatic and verification polynomials from the possible value of polynomials produced with simulated responses. To prove indistiguishability, we apply a standard 'hybrid' argument by producing a sequence of algorithms, replacing computation of responses by computation of commitments to constant coefficients, one at a time.

References

1. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. J. ACM 38(1), 691–729 (1991)
2. Kilian, J.: A note on efficient proofs and arguments. In: Proceedings of ACM STOC 1992 (1992)
3. Boyar, J., Brassard, G., Peralta, R.: Subquadratic zero-knowledge. J. ACM 42, 1169–1193 (1995)
4. Cramer, R., Damgård, I.: Linear zero-knowledge - a note on efficient zero-knowledge proofs and arguments. In: Proceedings of ACM STOC 1997, pp. 436–445 (1997)
5. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. SIAM Journal on Computing 18(1) (1989)
6. Goldreich, O., Krawczyk, H.: On the composition of zero-knowledge proof systems. SIAM Journal on Computing 22(6), 1163–1175 (1993)
7. Goldreich, O., Kahan, A.: How to construct constant-round zero-knowledge proof systems for NP. Journal of Cryptology 9(2), 167–189 (1996)
8. Schnorr, C.-P.: Efficient Identification and Signatures for Smart Cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (1990)
9. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge proofs from secure multiparty computation. SIAM J. Comput. 39(3), 1121–1152 (2009)
10. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM 39(4), 859–868 (1992)
11. Shamir, A.: IP=PSPACE. J. ACM 39(4), 869–877 (1992)
12. Fedyukovych, V.: An argument for Hamiltonicity. In: Conference on Mathematics and Inf. Tech. Security (MaBIT-2008), also Cryptology ePrint Archive, Report 2008/363 (2008)
13. Fedyukovych, V.: Protocols for graph isomorphism and hamiltonicity. In: Central European Conference on Cryptography (2009)

14. Micciancio, D., Petrank, E.: Simulatable Commitments and Efficient Concurrent Zero-Knowledge. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 140–159. Springer, Heidelberg (2003)
15. Cramer, R., Damgård, I.B.: Zero-Knowledge Proofs for Finite Field Arithmetic or: Can Zero-Knowledge Be for Free? In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 424–441. Springer, Heidelberg (1998)
16. Chaum, D., Evertse, J.-H., van de Graaf, J.: An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations. In: Price, W.L., Chaum, D. (eds.) EUROCRYPT 1987. LNCS, vol. 304, pp. 127–141. Springer, Heidelberg (1988)
17. Pedersen, T.P.: Non-interactive and Information-Theoretic Secure Verifiable Secret Sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
18. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM 27, 701–717 (1980)

Cluster Vertex Deletion: A Parameterization between Vertex Cover and Clique-Width[*]

Martin Doucha and Jan Kratochvíl

Department of Applied Mathematics, Faculty of Mathematics and Physics,
Charles University, Malostranské nám. 25, 118 00 Prague, Czech Republic
{doucm6am,honza}@kam.mff.cuni.cz

Abstract. The *cluster vertex deletion* number of a graph is the minimum number of its vertices whose deletion results in a disjoint union of complete graphs. This generalizes the vertex cover number, provides an upper bound to the clique-width and is related to the previously studied notion of the twin cover of the graph under consideration. We study the fixed parameter tractability of basic graph theoretic problems related to coloring and Hamiltonicity parameterized by cluster vertex deletion number. Our results show that most of these problems remain fixed parameter tractable as well, and thus we push the borderline between tractability and intractability towards the clique-width parameter.

1 Introduction

1.1 Parameterized Complexity

The theory of parameterized complexity, first introduced by Abrahamson et al. [1] and developed by Downey and Fellows in a series of research papers and summarized in their famous textbook [12], has become in recent years a widely used approach to understand subtle aspects of the complexity of computationally hard problems. If an instance of a problem comes naturally structured into two parts, one of size n with expected unlimited growth and the other one size k which becomes interesting from the practical point of view even for not too big values of k, the goal is to find an algorithm running in time $O(f(k)n^{O(1)})$ for any function f. If such an algorithm exists, the problem is called *Fixed Parameter Tractable*, FPT for short. Many problems allow more or less straightforward algorithms of running time $O(n^{g(k)})$ (maximum independent set in a graph being one of the most fundamental examples) and in such a case the goal is "to remove the dependency on k from the exponent". Downey and Fellows have developed a theory of fixed parameter tractability and intractability as a counterpart of the NP-completeness of Cook and Karp, the role of "hard" problems is played by a tower of $W[t]$ classes. For definitions and details the reader is referred to [12]. For the purpose of this paper we note that Clique and Independent Set are problems complete for $W[1]$, the first level of the hierarchy.

[*] Supported by Czech Research grants CE-ITI GAČR P202/12/6061 and GAUK 64110.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 348–359, 2012.

1.2 Parameterization by Width Parameters

The *tree-width* of a graph was introduced by Robertson and Seymour in their extensive work on graph minors as a graph invariant that measures how close is a given graph to a tree structure. However, in 1978 Arnborg et al. [4] had already defined the notion of *partial k-trees* as subgraphs of graphs that can be constructed from a k-clique by sequential addition of simplicial vertices of degree k (the so-called *k-trees*). It turns out that the tree-width of a graph G, denoted by $tw(G)$, is the minimum k such that G is a partial k-tree. The importance of partial k-trees from the computational point of view is that most polynomial time algorithms for trees can be extended to them. This has been noted already in [4] and expressed in the most elegant way by Courcelle [10] who proved that every graph property expressible in the Monadic Second Order Logic (MSOL) can be tested in linear time on partial k-trees (for every fixed k). From the parameterized complexity point of view this means that every MSOL expressible graph property is FPT when parameterized by the tree-width of the input graph. Determining the tree-width of a graph is NP-hard, but for any fixed k, graphs of tree-width at most k form a minor closed class of graphs, and hence are recognizable in polynomial time by the Robertson-Seymour graph minor machinery; an explicit linear time algorithm is due to Bodlaender [6]. The latter means that the problem of determining tree-width is FPT when parameterized by the tree-width of the input graph itself.

Courcelle and Olariu [11] have introduced another width parameter for graphs, the so-called *clique-width* which captures the complexity of constructing a given graph by certain operations that then allow algorithms to recurse along the lines of this construction. This notion generalizes tree-width in the sense that every graph of bounded tree-width has bounded clique-width as well [9], and thus the existence of polynomial time algorithms for graphs of bounded clique-width for various problems supersedes the results on tree-width. The computational aspects of clique-width itself are less understood than those of tree-width. Only recently it has been shown that determining clique-width is NP-hard [15], but the parameterized complexity of recognizing graphs of bounded clique-width is still open. Though many natural problems are polynomial time solvable for graphs of bounded clique-width, the exponents of the polynomial function describing the running times of best known algorithms depend on the width. It has been shown recently that at least for Hamiltonicity and coloring problems this cannot be avoided (unless FTP = W[1]), as these problems are W[1]-hard when parameterized by clique-width [19].

Fellows et al. [13] show that some variants of graphs coloring, namely Precoloring extension and Equitable coloring, are W[1]-hard when parameterized by tree-width. In another paper [14] a similar set of authors suggest the consideration of other parameters with respect to which hard problems may become fixed parameter tractable. A particularly suitable parameter seems to be the *vertex cover number* of a graph. A vertex cover is the complement of an independent set, i.e., a set of vertices that meets every edge of the graph (in at least one vertex). The vertex cover number of a graph G, denoted by $vc(G)$, is the smallest size of a vertex cover in G. It is easily seen that $tw(G) \leq vc(G)$ holds for every graph G, and thus every problem is at least as hard when parameterized by tree-width as it is when parameterized by vertex cover number. It is shown by Fiala et al. [16] that indeed Precoloring extension, Equitable coloring and other coloring related problems are FPT when parameterized by vertex cover number. The aim of this paper is to show that some of those results hold for a more general parameter.

1.3 Cluster Vertex Deletion

In this paper we initiate the study of parameterization by the *cluster vertex deletion* number of a graph. A cluster vertex deletion of a graph is a set of vertices whose deletion results in a disjoint union of complete graphs. The size of a smallest cluster vertex deletion of a graph G is denoted by $ucvd(G)$. If all the cliques have sizes at most c, where c is a positive integer, we call the deletion a *c-bounded cluster vertex deletion* of G. The smallest size of a c-bounded cluster vertex deletion of G is denoted by $bcvd_c(G)$. Obviously $ucvd(G) \leq bcvd_c(G) \leq vc(G)$ for every graph and $c \geq 1$. As we show later, the clique-width of a graph is bounded by a function of its cluster vertex deletion number and tree-width is bounded by a function of c-bounded cluster vertex deletion number. Thus studying the parameterized complexity of problems parameterized by the cluster vertex deletion number refines the study of the boundary between tractable and intractable parameterizations. Our findings about problems related to graph coloring and Hamiltonicity are listed in Table 1 below. The problems we have considered are rather common and therefore we postpone their formal definition to the particular sections dealing with their study.

It should be mentioned that our invariants are related to the *twin cover* introduced by Ganian in [20] as a subset of the vertex set of a graph under consideration such that every two adjacent vertices lying outside this subset are true twins, i.e., they have the same closed neighborhoods in the graph. It immediately follows that the vertices lying outside a twin cover must induce a disjoint union of complete graphs, and hence $ucvd(G) \leq tc(G)$ holds for any graph G, where $tc(G)$ denotes the size of a smallest twin cover in G.

Let us also note at this point that though the results mostly say that FPT algorithms for parameterization by vertex cover number can be extended to parameterization by cluster vertex deletion number, the proofs (namely in the case of Equitable Coloring) are novel compared to [16]. The new idea of using matchings (developed in parallel with Ganian's network flow technique for parameterization by twin cover) enabled the design of an FPT algorithm for the much more general problem Number Coloring.

1.4 Notations and Organization of the Paper

We consider finite simple undirected graphs, i.e., graphs without loops or multiple edges. Edges are viewed as two-element subsets of the vertex set, but for

Table 1. Parameterized complexity of the problems considered with respect to parameterizations by vertex cover number (VC), bounded cluster vertex deletion number (BCVD), twin cover number (TC), unbounded cluster vertex deletion number (UCVD), tree-width (TW), and clique-width (CW). Results proved in this paper are marked by asterisk *. Results not explicitly stated but straightforwardly following from statements proved in a reference [x] are marked by ←[x].

Problem	**VC**	**BCVD**	**TC**	**UCVD**	**TW**	**CW**
Precoloring Extension	FPT[16]	FPT*	FPT[20]	W[1]-hard*	W[1]-hard[13]	W[1]-hard←[13]
Number Coloring	FPT←*	FPT*	?	?	W[1]-hard←[13]	W[1]-hard←[13]
Equitable Coloring	FPT[16]	FPT*	FPT[20]	FPT*	W[1]-hard[13]	W[1]-hard←[13]
Chromatic Number	FPT←[3]	FPT←[3]	FPT[20]	FPT*	FPT[3]	W[1]-hard[18]
Hamiltonian Path	FPT←[3]	FPT←[3]	FPT←*	FPT*	FPT[3]	W[1]-hard[18]
Hamiltonian Cycle	FPT←[3]	FPT←[3]	FPT←*	FPT*	FPT[3]	W[1]-hard[18]

simplicity we write uv for the edge joining vertices u and v. Thus a graph is a pair $G = (V, E)$ where $V = V(G)$ is the vertex set of G and $E = E(G) \subseteq \binom{V}{2}$ is its edge set. We reserve n for the number of vertices and m for the number of edges of the graph under consideration (which usually will be G). Adjacent vertices are called *neighbors* and the set of neighbors of a vertex u is denoted by $N(u)$, and it is called the *open neighborhood* of u. The *closed neighborhood* of u is $N[u] = N(u) \cup \{u\}$. If $W \subseteq V$ is a subset of the vertex set of G, then $G[W]$ denotes the subgraph of G *induced* by W. A set A of vertices is *independent* if $G[A]$ is edgeless, and it is a *clique* if $G[A]$ is a complete graph. A subset A of vertices is called a *vertex cover* if $V \setminus A$ is an independent set. A *coloring* of G is any mapping from the vertex set to a set (usually referred to as the set of colors). A coloring is *proper* if adjacent vertices are mapped onto different colors, i.e., if the preimage of every color is an independent set.

We assume that the reader is familiar with the notions of *tree-width* and *clique-width*. For the definition of the perhaps less well known notion of *twin cover* we refer to [20].

In Section 2 we recollect the definition of cluster vertex deletion and the relationships to other width parameters. Sections 3–6 are devoted to the study of the parameterized complexity of the problems Precoloring Extension, Equitable Coloring, Number Coloring, Chromatic Number, and Hamiltonian Cycle and Path. In Section 7 we list open problems that we find of particular interest.

2 Properties of Cluster Vertex Deletion

In this section we restate the definitions of bounded and unbounded cluster vertex deletion in a formal way, comment on their computational complexities and study their relationships to other parameters relevant for this paper.

Definition 1 (Unbounded and c-Bounded Cluster Vertex Deletion). *An* (unbounded) cluster vertex deletion *in a graph G is a set $W \subseteq V(G)$ of vertices such that $G \setminus W$ is a disjoint union of cliques. The minimum size of such a W is denoted by $ucvd(G)$.*
A cluster vertex deletion W of G is called a c-bounded cluster vertex deletion *if each connected component of $G \setminus W$ has size at most c. The minimum size of such a W is denoted by $bcvd_c(G)$. For the sake of brevity we call a c-bounded cluster vertex deletion of size at most k a* (k, c)-deletion *of G.*

The cluster vertex deletion problem has been studied in [22] and [17] in relationship to cluster editing, a problem whose motivation is coming from computational biology or relational structures. As noted in these papers, the cluster vertex deletion problem is NP-complete, since disjoint unions of cliques are characterized by forbidden P_3, i.e. the path of length two, as an induced subgraph. A classical result of Lewis and Yannakakis [24] says that minimum vertex deletion problems to hit a hereditary class of graphs are all NP-complete. It immediately follows that also deciding $bcvd_c(G) \leq k$ is NP-complete for every c, since the class of disjoint unions of cliques of size at most c is characterized by two forbidden induced subgraphs – P_3 and K_{c+1}, the complete graph on $c + 1$ vertices.

We are introducing the bounded version of cluster vertex deletion with the argument that in the biological sampling application, typical examples will have equivalence classes of the samples of bounded size. It is therefore interesting to study the complexity of standard optimization problems with respect to the bounded case as well.

The first task is of course to explore the parameterized complexity of these problems themselves. Hüffner et al. [22] show that the cluster vertex deletion of a graph can be found in time $O(2^k k^6 \log k + nm)$, and hence the problem is in FPT when parameterized by the cluster vertex deletion number. We have a similar result for the bounded version.

Theorem 1. *If a graph G has a (k, c)-deletion, then it can be found in time $O(n+m+(k+k(k+c)c)^{k+2})$. Hence c-Bounded Cluster Vertex Deletion is FPT when parameterized by both c and the size of the deletion.*

Proof. Suppose G has a (k, c)-deletion W. Then the vertices outside W have degrees less than $k+c$. This suggests the first reduction rule – remove all vertices of degree $k + c$ or higher from G, put them into the deletion and reduce k to k' by subtracting the number of the removed vertices. The second reduction rule is to remove vertices which belong to clique components of order at most c (they can be left outside of W for no penalty).

The remaining graph G' has maximum degree less than $k + c$ and every component of $G' \setminus W$ is adjacent to at least one vertex of W. Thus G' has at most $k + k(k + c)c$ vertices. We then find a (k', c)-deletion of G' using brute force in time $O((k + (k + c)c)^{k+2})$ by processing all of its k-element subsets as candidates for $W \cap V(G')$ and checking if their deletion leaves a disjoint union of small cliques. □

Our aim is to compare the strengths of parameterizations by different parameters. Towards this end we need to compare when boundedness of one parameter implies the same of the other. It is well known [9] that bounded tree-width implies bounded clique-width. The following observation follows straightforwardly from the definitions of vertex cover and twin cover numbers of a graph G (denoted by $vc(G)$ and $tc(G)$, respectively).

Observation 2. *For every graph G and every positive integer c, it holds that $ucvd(G) \leq tc(G) \leq vc(G)$ and $ucvd(G) \leq bcvd_c(G) \leq vc(G)$.*

The relationships of bounded and unbounded cluster vertex deletion numbers to tree-width and clique-width are described by the following theorem.

Theorem 3. *For every graph G and every positive integer c, the following hold*
i) $tw(G) \leq bcvd_c(G) + c - 1$ and
ii) $cwd(G) \leq ucvd(G) + 3$.

Proof. i) Let W be the deletion in G, $|W| \leq k$, and let $C_1,\ldots,C_t$ be the cliques of $G \setminus W$. A suitable tree-decomposition is a path of length $t - 1$ with vertices assigned to bags $C_i \cup W$.

ii) Let W be the deletion in G, $|W| \leq k$. We begin the construction of G by creating vertices of W, each with its own unique label, and connecting them with edges as needed. We have used at most k labels.

We use three more labels: label $k + 1$ for finished components of $G \setminus W$, label $k + 2$ for components under construction and label $k + 3$ for an active vertex. Then we build the components of $G \setminus W$ one by one: If there are no vertices left in the current component under construction, we relabel $k + 2$ to $k + 1$ and move on to the next component. If there are vertices left, we choose one as the active vertex, create a new vertex for it with label $k + 3$ and add it to the already constructed part of G. Then we connect the new vertex (the only vertex with label $k + 3$) to vertices with label $k + 2$ and relevant vertices in W. Finally, we relabel $k + 3$ to $k + 2$ and continue with the next vertex or component as described above. □

The relationships between the parameters are illustrated in a Hasse-like diagram in Figure 2. Its meaning should be understood as follows. If an invariant A is above an invariant B and a problem P is FPT when parameterized by B, then P is also FPT when parameterized by A (and vice versa for W[1]-hardness). With this picture in mind we move to the parameterized complexity investigation of concrete problems in the next sections.

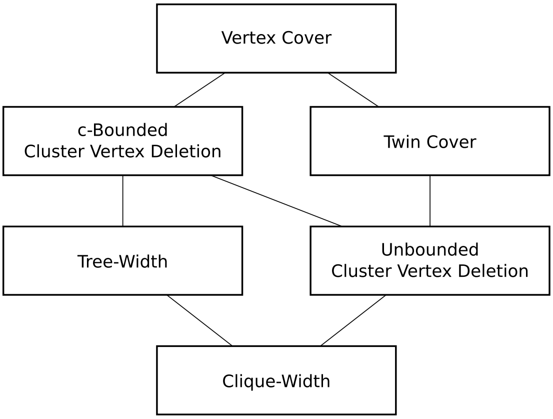

Vertex Cover
c-Bounded Cluster Vertex Deletion
Twin Cover
Tree-Width
Unbounded Cluster Vertex Deletion
Clique-Width

Fig. 1. The Hasse-like diagram of graph parameterization relationships

3 Precoloring Extension

In this section we consider the first variant of graph coloring. Note that the second problem, List Coloring, is only presented as an auxiliary problem for use in proofs. We do not list it in Table 1 since it is W[1]-hard already when parameterized by vertex cover number [14]. Also note that Precoloring Extension is the only one of the problems considered where we know that the tractability differs when parameterized by bounded and unbounded cluster vertex deletion numbers.

Precoloring Extension
Input: A graph G, the number of colors $r \in \mathbb{N}$, a set $X \subseteq V(G)$, and a precoloring $p : X \to \{1, \ldots, r\}$.
Question: Does there exist a proper coloring $q : V(G) \to \{1, \ldots, r\}$ of G such that $q(u) = p(u)$ for every $u \in X$?

List Coloring
Input: A graph G, the number of colors $r \in \mathbb{N}$, and an assignment $L : V(G) \to 2^{\{1,\ldots,r\}}$ of color lists to the vertices of G.
Question: Does there exist a proper coloring $q : V(G) \to \{1, \ldots, r\}$ of G such that $q(u) \in L(u)$ for every $u \in V(G)$?

3.1 Parameterization by Bounded Cluster Vertex Deletion Number

Theorem 4. Precoloring Extension *can be solved in time* $O((k+c)^{k+2}cnr) = O((k+c)^{k+2}cn^2)$ *on graphs with a* (k, c)*-deletion.*

Proof. Let W be a (k, c)-deletion in G. We begin by assigning lists of colors to all unprecolored vertices as permitted by the precoloring of their neighbors. Then we remove the precolored vertices, thus transforming G into an instance G', L of List coloring. This takes $O(n(k + r + c))$ time.

If $r \geq k+c$, each proper coloring of W can be extended to the rest of G' using a greedy algorithm. Similarly, if some vertex $u \in W$ has at least $|W|$ colors in its list, any proper coloring of $W \setminus \{u\}$ can be extended to a proper coloring of W. We use this observation to build a sequence of subsets of W. Let $W_0 = W$ be the first set in the sequence. Set $W_i = W_{i-1} \setminus \{v_i\}$ if there is a vertex $v_i \in W_{i-1}$ with at least $|W_{i-1}|$ colors in its list. Let W' be the last set in the sequence. Obviously, each vertex in W' has less than $|W'| \leq k$ colors in its list. We can therefore try all possible colorings of W' (at most k^k possible combinations) and if we find a proper coloring, we can extend it to the rest of G' following the lines explained above. All of these can be performed in time $O(nrk^k)$.

If $r < k+c$, we can (and must) try all possible colorings of W (at most $(k+c)^k$ combinations). Then we try to extend each proper coloring of W to the rest of G' by finding a matching in a bipartite auxiliary graph H. Before we construct this graph, we first update the color lists according to the coloring of W. The auxiliary graph H will consist of connected components representing the connected components of $G' \setminus W$. Given a component C of $G' \setminus W$, one part of H will consist of vertices representing the colors in the union of all color lists in C. The other part will represent individual vertices of C. Each vertex in the second part will be connected to the vertices representing colors in the color list of its corresponding vertex in G'. Obviously, a proper coloring of W can be extended to the rest of G' if and only if there is a matching fully satisfying the second part of H. Since each component of the auxiliary graph has at most $c+r$ vertices and at most cr edges, deciding the existence of a suitable matching takes time $O(ncr\sqrt{c+r}) = O(nc(k+c)^{3/2})$. Updating the lists for each coloring of W takes $O(n(k+c))$ time. □

3.2 Parameterization by Unbounded Cluster Vertex Deletion Number

Theorem 5. Precoloring extension *parameterized by the unbounded cluster vertex deletion number of the input graph is W[1]-hard.*

Proof. We prove the W[1]-hardness by reducing from List Coloring parameterized by vertex cover number of G which is known to be W[1]-hard [13],[14]. Given an instance of List coloring where G has a vertex cover of size k, let $C = \{1, \ldots, r\}$ be the set of all colors.

We construct the instance of Precoloring extension G', p as follows: For each vertex $v \in V(G)$, we adjoin to v a new complete graph on $|C \setminus L(v)|$ vertices precolored by all colors in $C \setminus L(v)$. Thus G' has a cluster vertex deletion of size k (the same vertices as in the vertex cover of G) with the precolored vertices serving as equivalent replacement for color lists. □

4 Equitable and Number Colorings

In this section we prove two of the main results of the paper. First we prove that Equitable Coloring is FPT even when parameterized by unbounded cluster vertex deletion number. But on the way actually prove another strengthening of previously known results – the fixed parameter tractability (when parameterized by bounded cluster vertex deletion number) for a more general problem Number Coloring which prescribes the sizes of color sets arbitrarily. The main technical tool is Lemma 1 which is then used in several situations. First we list the problems under consideration.

Equitable Coloring
Input: A graph G and the number of colors $r \in \mathbb{N}$.
Question: Does G allow a proper coloring $q : V(G) \to \{1, \ldots, r\}$ such that $||q^{-1}(i)| - |q^{-1}(j)|| \leq 1$ holds for all $i, j \in \{1, \ldots, r\}$?

Number Coloring
Input: A graph G, the number of colors $r \in \mathbb{N}$ and positive integers $n_1, \ldots, n_r \in \mathbb{N}$ such that $\sum_{i=1}^{r} n_i = |V(G)|$.
Question: Does G allow a proper coloring $q : V(G) \to \{1, \ldots, r\}$ such that $|q^{-1}(i)| = n_i$ for all $i \in \{1, \ldots, r\}$?

Number List Coloring
Input: A graph G, the number of colors $r \in \mathbb{N}$, positive integers $n_1, \ldots, n_r \in \mathbb{N}$ such that $\sum_{i=1}^{r} n_i = |V(G)|$, and an assignment of lists $L : V(G) \to 2^{\{1,\ldots,r\}}$.
Question: Does G allow a proper coloring $q : V(G) \to \{1, \ldots, r\}$ such that $|q^{-1}(i)| = n_i$ for all $i \in \{1, \ldots, r\}$ and $q(u) \in L(u)$ for all $u \in V(G)$?

Lemma 1. *Given an instance of* Number List Coloring *on a graph G such that $G[i]$ (the subgraphs of G induced by the vertices containing color i in their lists) is a disjoint union of cliques for each $i \in \{1, \ldots, r\}$, the problem can be solved in time $O(n^{\frac{5}{2}} r^{\frac{3}{2}})$.*

For space limitations the proof will appear in the full version only.

4.1 Parameterization by Unbounded Cluster Vertex Deletion Number

Theorem 6. *Given an instance of* Number Coloring *on a graph G such that $ucvd(G) \leq k$ and such that the colors can be sorted into s groups, each group containing colors with the same value of n_i, the problem can be solved in time $O((ks)^k n^{\frac{5}{2}} r^{\frac{3}{2}})$.*

Proof. Let W be cluster vertex deletion in G of size at most k. We choose k representatives from each group (or the entire group if it has less than k colors) and try all $(ks)^k$ possible colorings of W using only the representative colors. We thus try all possible colorings up to renaming of colors in the same group.

For each of these colorings we check whether they can be extended into a correct solution. Given a proper coloring of W that does not exceed any n_i, we assign color lists to vertices of $G \setminus W$ according to their neighborhoods in W. We also adjust the values n_i by the numbers of vertices in W colored by each color. In this way we construct an instance of Number List Coloring which we solve using Lemma 1, since indeed every color induces a disjoint union of cliques. □

Corollary 1. Equitable Coloring *is FPT when parameterized by unbounded cluster vertex deletion number (and hence also when parameterized by both c and $bcvd_c(G)$).*

Proof. Equitable Coloring is a special case of Number Coloring with two groups of colors with the same values n_i within the groups (namely $\lfloor \frac{n}{r} \rfloor$ and $\lceil \frac{n}{r} \rceil$). □

4.2 Parameterization by Bounded Cluster Vertex Deletion Number

Theorem 7. Number coloring *can be solved on graphs with a (k, c)-deletion in time $O((2k^2 + k + c)^k n^{\frac{5}{2}} r^{\frac{3}{2}})$.*

For space limitations the proof will appear in the full version only.

5 Chromatic Number

In this section we deal with the most standard variant of a coloring problem. Since it is well known to be FPT when parameterized by tree-width, we only deal with parameterization by unbounded cluster vertex deletion number. Due to space limitations the proof will appear in the full version only.

Chromatic Number
Input: A graph G and an integer r.
Question: Does G allow a proper coloring $p : V(G) \to \{1, \dots, r\}$?

Theorem 8. Chromatic Number *can be solved in time $O((\frac{0.792k}{\ln(k+1)})^k (kn)^{\frac{3}{2}})$ for a graph G with $ucvd(G) \leq k$.*

6 Hamiltonian Path and Cycle

In this section we consider problems that are of different nature than the variants of graph coloring that we have delt with so far. Still Hamiltonicity question belong to the basic graph problems and we could not resist including them in our report. Again we show FPT for parameterization by unbounded cluster vertex deletion number and thus do not have to deal with the bounded case (where FPT also follows from parameterization by tree-width). For space limitations the proof will appear in the full version only.

Hamiltonian Path
Input: A graph G.
Question: Does there exist a permutation π of the vertices of G such that for each $1 \leq i \leq n - 1$ it holds that $v_{\pi(i)} v_{\pi(i+1)} \in E(G)$?

Hamiltonian Cycle
Input: A graph G.
Question: Does there exist an permutation π which defines a Hamiltonian path and such that $v_{\pi(1)} v_{\pi(n)} \in E(G)$?

Theorem 9. Hamiltonian Path *can be solved in time* $O(k!(k+1)!\binom{\lceil \frac{k+1}{2} \rceil^2}{k+1}\binom{4k^2}{2k}n)$ *on graphs* G *such that* $ucvd(G) \leq k$.

Theorem 10. Hamiltonian Cycle *can be solved in time* $O((k-1)!k!\binom{\lceil \frac{k}{2} \rceil^2}{k}\binom{4k^2}{2k}n)$ *on graphs* G *such that* $ucvd(G) \leq k$.

Proof. We prove the theorem using a slightly modified algorithm for Hamiltonian Path. Let W be a cluster vertex deletion of G of size at most k and let $u \in W$ be any vertex. We remove u from G (u will be the first and also the last vertex of the cycle cut into a path) and call the algorithm for Hamiltonian Path which will do one extra test in the final step to see if there are suitable edges between u and the first and last component of the path. □

7 Conclusion

We have initiated the study of the fixed parameter tractability or intractability of several coloring and Hamiltonicity problems when parameterized by two graph invariants generalizing the vertex cover number. In one case we found an FPT algorithm for a rather strong question Number Coloring. We believe that the complexity of this problem deserves further attention:

Problem. What is the parameterized complexity of Number Coloring parameterized by (unbounded) cluster vertex deletion number and by twin cover number?

Our second open question is actually a question of Henning Fernau that has inspired our research. Henning suggested in a 2009 personal communication to study parameterization by "Bounded Component Vertex Cover", i.e. by the minimum size of a set of vertices whose removal leaves all components of the graph under consideration of bounded size. Even size 3 (allowing components isomorphic to $K_{1,2}$) seems quite hard for Equitable Coloring and similar questions. We would like to use this opportunity to thank Henning for a stimulating question.

References

1. Abrahamson, K.R., Ellis, J.A., Fellows, M.R., Mata, M.E.: On the Complexity of Fixed Parameter Problems (Extended Abstract). In: FOCS 1989, pp. 210–215 (1989)
2. Abu-Khzam, F.N.: A kernelization algorithm for d-Hitting Set. J. Comput. Syst. Sci. 76(7), 524–531 (2010)
3. Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees. Discrete Applied Mathematics 23(1), 11–24 (1989)
4. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of Finding Embeddings in a k-Tree. SIAM. J. on Algebraic and Discrete Methods 8, 277–284 (1978)

5. Berend, D., Tassa, T.: Improved bounds on bell numbers and on moments of sums of random variables. Probability and Mathematical Statistics 30, 185–205 (2010)
6. Bodlaender, H.L.: A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. SIAM J. Comput. 25, 1305–1317 (1996)
7. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Information Processing Letters 58(4), 171–176 (1996)
8. Chen, J., Kanj, I.A., Xia, G.: Improved Parameterized Upper Bounds for Vertex Cover. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 238–249. Springer, Heidelberg (2006)
9. Corneil, D.G., Rotics, U.: On the relationship between clique-width and treewidth. SIAM J. Comput. 34, 825–847 (2005)
10. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Information and Computation 85, 12–75 (1990)
11. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Disc. Appl. Math. 101, 77–114 (2000)
12. Downey, R.G., Fellows, M.R.: Parameterized complexity. Monographs in Computer Science. Springer (1999)
13. Fellows, M.R., Fomin, F.V., Lokshtanov, D., Rosamond, F., Saurabh, S., Szeider, S., Thomassen, C.: On the complexity of some colorful problems parameterized by treewidth. Information and Computation 209, 143–153 (2011)
14. Fellows, M.R., Lokshtanov, D., Misra, N., Rosamond, F.A., Saurabh, S.: Graph Layout problems Parameterized by Vertex Cover. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 294–305. Springer, Heidelberg (2008)
15. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-width is NP-complete. SIAM J. Discr. Math. 23(2), 909–939 (2009)
16. Fiala, J., Golovach, P.A., Kratochvíl, J.: Parameterized Complexity of Coloring Problems: Treewidth versus Vertex Cover (Extended Abstract). In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 221–230. Springer, Heidelberg (2009)
17. Fomin, F.V., Gaspers, S., Kratsch, D., Liedloff, M., Saurabh, S.: Iterative compression and exact algorithms. Theor. Comput. Sci. 411(7-9), 1045–1053 (2010)
18. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Clique-width: On the Price of Generality. In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 825–834 (2009)
19. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Intractability of Clique-Width Parameterizations. SIAM J. Comput. 39(5), 1941–1956 (2010)
20. Ganian, R.: Twin-Cover: Beyond Vertex Cover in Parameterized Algorithmics. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 259–271. Springer, Heidelberg (2012)
21. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM J. Comput. 2, 225–231
22. Hüffner, F., Komusiewicz, C., Moser, H., Niedermeier, R.: Fixed-Parameter Algorithms for Cluster Vertex Deletion. Theory Comput. Syst. 47(1), 196–217 (2010)
23. Karp, R.M.: Reducibility Among Combinatorial Problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations: Proc. of a Symp. on the Complexity of Computer Computations. The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York (1972)
24. Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is NP-complete. Journal of Computer and System Sciences 20(2), 219–230 (1980)

On the Impact of Fair Best Response Dynamics*

Angelo Fanelli[1], Luca Moscardelli[2], and Alexander Skopalik[3]

[1] Division of Mathematical Sciences, School of Physical and Mathematical Sciences,
Nanyang Technological University, Singapore
angelo.fanelli@ntu.edu.sg

[2] Department of Economic Studies, University of Chieti-Pescara, Italy
luca.moscardelli@unich.it

[3] Department of Computer Science, TU Dortmund, Germany
alexander.skopalik@tu-dortmund.de

Abstract. In this work we completely characterize how the frequency with which each player participates in the game dynamics affects the possibility of reaching efficient states, i.e., states with an approximation ratio within a constant factor from the price of anarchy, within a polynomially bounded number of best responses. We focus on the well known class of linear congestion games and we show that (i) if each player is allowed to play at least once and at most β times in T best responses, states with approximation ratio $O(\beta)$ times the price of anarchy are reached after $T\lceil\log\log n\rceil$ best responses, and that (ii) such a bound is essentially tight also after exponentially many ones. One important consequence of our result is that the fairness among players is a necessary and sufficient condition for guaranteeing a fast convergence to efficient states. This answers the important question of the maximum order of β needed to fast obtain efficient states, left open by [10,11] and [3], in which fast convergence for constant β and very slow convergence for $\beta = O(n)$ have been shown, respectively. Finally, we show that the structure of the game implicitly affects its performances. In particular, we prove that in the symmetric setting, in which all players share the same set of strategies, the game always converges to an efficient state after a polynomial number of best responses, regardless of the frequency each player moves with. All the results extend to weighted congestion games.

Keywords: Congestion Games, Speed of Convergence, Best Response Dynamics.

1 Introduction

Congestion games are used for modelling non-cooperative systems in which a set of resources are shared among a set of selfish players. In a congestion game we

* This research was partially supported by the grant NRF-RF2009-08 "Algorithmic aspects of coalitional games", by the PRIN 2008 research project COGENT (COmputational and GamE-theoretic aspects of uncoordinated NeTworks), funded by the Italian Ministry of University and Research, and by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project A2.

have a set of m resources and a set of n players. Each player's strategy consists of a subset of resources. The delay of a particular resource e depends on its congestion, corresponding to the number of players choosing e, and the cost of each player i is the sum of the delays associated with the resources selected by i. In this work we focus on linear congestion games where the delays are linear functions. A congestion game is called symmetric if all players share the same strategy set. A state of the game is any combination of strategies for the players and its social cost, defined as the sum of the players' costs, denotes its quality from a global perspective. The social optimum denotes the minimum possible social cost among all the states of the game.

Related Work. Rosenthal [14] has shown, by a potential function argument, that the natural decentralized mechanism known as Nash dynamics consisting in a sequence of moves in which at each one some player switches its strategy to a better alternative, is guaranteed to converge to a pure Nash equilibrium [13].

In order to measure the degradation of social welfare due to the selfish behavior of the players, Koutsoupias and Papadimitriou [12] defined the price of anarchy as the worst-case ratio between the social cost in a Nash equilibrium and that of a social optimum. The price of anarchy for congestion games has been investigated by Awerbuch et al. [2] and Christodoulou and Koutsoupias [6]. They both proved that the price of anarchy for congestion games with linear delays is 5/2.

The existence of a potential function relates the class of congestion games to the class of polynomial local search problems (PLS) [8]. Fabrikant et al. [9] proved that, even for symmetric congestion games, the problem of computing Nash equilibria is PLS-complete [8]. One major consequence of the completeness result is the existence of congestion games with initial states such that any improvement sequence starting from these states needs an exponential number of steps in the number of players n in order to reach a Nash equilibrium. A recent result by Ackermann et al. [1] shows that the previous negative result holds even in the restricted case of congestion games with linear delay functions.

The negative results on computing equilibria in congestion games have lead to the development of the concept of ϵ-Nash equilibrium, in which no player can decrease its cost by a factor of more than ϵ. Unfortunately, as shown by Skopalik and Vöcking [15], also the problem of finding an ϵ-Nash equilibrium in congestion games is PLS-complete for any ϵ, though, under some restrictions on the delay functions, Chien and Sinclair [5] proved that in symmetric congestion games the convergence to ϵ-Nash equilibrium is polynomial in the description of the game and the minimal number of steps within which each player has a chance to move.

Since negative results tend to dominate the issues relative to the complexity of computing equilibria, another natural arising question is whether efficient states (with a social cost comparable to the one of any Nash equilibrium) can be reached by best response moves in a reasonable amount of time (e.g., [3,7,10,11]). We measure the efficiency of a state by the ratio among its cost and the optimal one, and we refer to it as the approximation ratio of the state. We generally say

that a state is efficient when its approximation ratio is within a constant factor from the price of anarchy. Since the price of anarchy of linear congestion games is known to be constant [2,6], efficient states approximate the social optimum by a constant factor. While Bilò et al. [4] considered such a problem restricted to the case in which the dynamics start from an empty state, proving that in such a setting an efficient state can be reached by allowing each player to move exactly once, we focus on the more general setting in which the dynamics start from a generic state. It is worth noticing that in the worst case, a generic Nash dynamics starting from an arbitrary state could never reach a state with an approximation ratio lower than the price of anarchy. Furthermore, by a potential function argument it is easy to show that in a linear congestion game, once a state S with a social cost $C(S)$ is reached, even if such a state is not a Nash equilibrium, we are guaranteed that for any subsequent state S' of the dynamics, $C(S') = O(C(S))$.
Awerbuch et al. [3] have proved that for linear congestion games, sequences of moves reducing the cost of each player by at least a factor of ϵ, converge to efficient states in a number of moves polynomial in $1/\epsilon$ and the number of players, under the minimal liveness condition that every player moves at least once every polynomial number of moves. Under the same liveness condition, they also proved that exact best response dynamics can guarantee the convergence to efficient states only after an exponential number of best responses [3]. Nevertheless, Fanelli et al. [10] have shown that, under more restrictive condition that each player plays exactly once every n best responses, any best response dynamics converges to an efficient state after $\Theta(n \log \log n)$ best responses. Subsequently, Fanelli and Moscardelli [11] extended the previous results to the more general case in which each player plays a constant number of times every $O(n)$ best responses.

Our Contribution. In this work we completely characterize how the frequency with which each player participates in the game dynamics affects the possibility of reaching efficient states. In particular, we close the most important open problem left open by [3] and [10,11] for linear congestion games. On the one hand, in [3] it is shown that, even after an exponential number of best responses, states with a very high approximation ratio, namely $\Omega\left(\frac{\sqrt{n}}{\log n}\right)$, can be reached. On the other hand, in [10,11] it is shown that, under the minimal liveness condition in which every player moves at least once every T steps, if players perform best responses such that each player is allowed to play at most $\beta = O(1)$ times any T steps (notice that $\beta = O(1)$ implies $T = O(n)$), after $T\lceil \log \log n \rceil$ best responses a state with a constant factor approximation ratio is reached.

The more β increases, the less the dynamics is fair with respect to the chance every player has of performing a best response: β measures the degree of unfairness of the dynamics. The important left open question was that of determining the maximum order of β needed to obtain fast convergence to efficient states: We answer this question by proving that, after $T\lceil \log \log n \rceil$ best responses, the dynamics reaches states with an approximation ratio of $O(\beta)$. Such a result is essentially tight since we are also able to show that, for any $\epsilon > 0$, there exist congestion games for which, even for an exponential number of best responses, states with an approximation ratio of $\Omega(\beta^{1-\epsilon})$ are obtained. Therefore, β constant as assumed in [10,11] is not only sufficient, but also necessary in order to reach efficient states after a polynomial number of best responses.

Finally, in the special case of symmetric congestion games, we show that the unfairness in best response dynamics does not affect the fast convergence to efficient states; namely, we prove that, for any β, after $T\lceil \log\log n \rceil$ best responses efficient states are always reached.

The paper is organized as follows: In the next section we provide the basic notation and definitions. Section 3 is devoted to the study of generic linear congestion games, while Section 4 analyzes the symmetric case. Finally, Section 5 provides some extensions of the results and gives some conclusive remarks.

2 Model and Definitions

A *congestion game* $\mathcal{G} = (N, E, (\Sigma_i)_{i\in N}, (f_e)_{e\in E}, (c_i)_{i\in N})$ is a non-cooperative strategic game characterized by the existence of a set E of resources to be shared by n players in $N = \{1, \ldots, n\}$.

Any strategy $s_i \in \Sigma_i$ of player $i \in N$ is a subset of resources, i.e., $\Sigma_i \subseteq 2^E$. A congestion game is called *symmetric* if all players share the same set of strategies, i.e., $\Sigma = \Sigma_i$ for every $i \in N$. Given a state or strategy profile $S = (s_1, \ldots, s_n)$ and a resource e, the number of players using e in S, called the congestion on e, is denoted by $n_e(S) = |\{i \in N \mid e \in s_i\}|$. A delay function $f_e : \mathbb{N} \mapsto \mathbb{Q}^+$ associates to resource e a delay depending on the congestion on e, so that the cost of player i for the pure strategy s_i is given by the sum of the delays associated with the resources in s_i, i.e., $c_i(S) = \sum_{e\in s_i} f_e(n_e(S))$.

In this paper we will focus on linear congestion games, that is having linear delay functions with nonnegative coefficients. More precisely, for every resource $e \in E$, $f_e(x) = a_e x + b_e$ for every resource $e \in E$, with $a_e, b_e \in \mathbb{Q}^+$.

Given the strategy profile $S = (s_1, \ldots, s_n)$, the social cost $C(S)$ of a given state S is defined as the sum of all the players' costs, i.e., $C(S) = \sum_{i\in N} c_i(S)$. An optimal strategy profile $S^* = (s_1^*, \ldots, s_n^*)$ is one having minimum social cost; we denote $C(S^*)$ by $\textsc{Opt}$. The *approximation ratio* of state S is given by the ratio between the social cost of S and the social optimum, i.e., $\frac{C(S)}{\textsc{Opt}}$. Moreover, given the strategy profile $S = (s_1, s_2, \ldots, s_n)$ and a strategy $s_i' \in \Sigma_i$, let $(S_{-i}, s_i') = (s_1, s_2, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_n)$, i.e., the strategy profile obtained from S if player i changes its strategy from s_i to s_i'.

The potential function is defined as $\Phi(S) = \sum_{e\in E} \sum_{j=1}^{n_e(S)} f_e(j)$. It is an *exact* potential function since it satisfies the property that for each player i and each strategy $s_i' \in \Sigma_i$ of i in S, it holds that $c_i(S_{-i}, s_i') - c_i(S) = \Phi(S_{-i}, s_i') - \Phi(S)$. It is worth noticing that in linear congestion games, for any state S, it holds $\Phi(S) \le C(S) \le 2\Phi(S)$.

Each player acts selfishly and aims at choosing the strategy decreasing its cost, given the strategy choices of other players. A *best response* of player i in S is a strategy $s_i^b \in \Sigma_i$ yielding the minimum possible cost, given the strategy choices of the other players, i.e., $c_i(S_{-i}, s_i^b) \leq c_i(S_{-i}, s_i')$ for any other strategy $s_i' \in \Sigma_i$. Moreover, if no $s_i' \in \Sigma_i$ is such that $c_i(S_{-i}, s_i') < c_i(S)$, the best response of i in S is s_i. We call a *best response dynamics* any sequence of best responses.

Given a best response dynamics starting from an arbitrary state, we are interested in the social cost of its final state. To this aim, we must consider dynamics in which each player performs a best response at least once in a given number T of best responses, otherwise one or more players could be "locked out" for arbitrarily long and we could not expect to bound the social cost of the state reached at the end of the dynamics. Therefore, we define a T-*covering* as a dynamics of T consecutive best responses in which each player moves at least once. More precisely, a T-covering $R = \left(S_R^0, \ldots, S_R^T\right)$ is composed of T best responses; S_R^0 is said to be the *initial* state of R and S_R^T is its *final* state. For every $1 \leq t \leq T$, let $\pi_R(t)$ be the player performing the t-th best response of R; π_R is such that every player performs at least a best response in R. In particular, for every $1 \leq t \leq T$, $S_R^t = \left((S_R^{t-1})_{-\pi_R(t)}, s'_{\pi_R(t)}\right)$ and $s'_{\pi_R(t)}$ is a best response of player $\pi_R(t)$ to S_R^{t-1}. For any $i = 1, \ldots, n$, the last best response performed by player i in R is the $\text{last}_R(i)$-th best response of R, leading from state $S^{\text{last}_R(i)-1}$ to state $S^{\text{last}_R(i)}$. When clear from the context, we will drop the index R from the notation, writing S^i, π and $\text{last}(i)$ instead of S_R^i, π_R and $\text{last}_R(i)$, respectively.

Definition 1 (T-Minimum Liveness Condition). *Given any $T \geq n$, a best response dynamics satisfies the T-Minimum Liveness Condition if it can be decomposed into a sequence of consecutive T-coverings.*

In Section 3.2 we show that (for the general asymmetric case) under such a condition the quality of the reached state can be very bad even considering $T = O(n)$ (see Corollary 1): It is worth noticing that in the considered congestion game, only $\sqrt[4]{n}$ players perform a lot of best responses ($\sqrt{n}$ best responses) in each covering, while the remaining $n - \sqrt[4]{n}$ players perform only one best response every T-covering. The idea here is that there is a sort of unfairness in the dynamics, given by the fact that the players do not have the same chances of performing best responses.

In order to quantify the impact of fairness on best response dynamics, we need an additional parameter β and we define a β-bounded T-covering as a T-covering in which every player performs at most β best responses.

Definition 2 ((T, β)-Fairness Condition). *Given any positive integers β and T such that $n \leq T \leq \beta \cdot n$, a dynamics satisfies the (T, β)-Fairness Condition if it can be decomposed into a sequence of consecutive β-bounded T-coverings.*

Notice that β is a sort of (un)fairness index: If β is constant, it means that every player plays at most a constant number of times in each T-covering and therefore the dynamics can be considered *fair.*

In order to prove our upper bound results, we will focus our attention on particular congestion games to which any linear congestion game is best-response reducible. The following definition formally states such a notion of reduction.

Definition 3 (Best-Response Reduction). *A congestion game $\mathcal{G}$ is Best-Response reducible to a congestion game $\mathcal{G}'$ with the same set of players if there exists an injective function g mapping any strategy profile S of $\mathcal{G}$ to a strategy profile $g(S)$ of $\mathcal{G}'$ such that*

(i) for any $i = 1, \dots, n$ the cost of player i in S is equal to the one of player i in $g(S)$

(ii) for any $i = 1, \dots, n$, there exists, in the game $\mathcal{G}$, a best response of player i in S leading to state S' if and only if there exists, in the game $\mathcal{G}'$, a best response of player i in $g(S)$ leading to state $g(S')$.

3 Asymmetric Congestion Games

In this section we first (in Subsection 3.1) provide an upper bound to the approximation ratio of the states reached after a dynamics satisfying the (T, β)-Minimum Liveness Condition, starting from an arbitrary state and composed by a number of best responses polynomial in n. Finally (in Subsection 3.2), we provide an almost matching lower bound holding for dynamics satisfying the same conditions.

3.1 Upper Bound

All the results hold for linear congestion games having delay functions $f_e(x) = a_e x + b_e$ with $a_e, b_e \geq 0$ for every $e \in E$. Since our bounds are given as a function of the number of players, as shown in [10], the following proposition allows us to focus on congestion games with identical delay functions $f(x) = x$.

Proposition 1 ([10]). *Any linear congestion game is best-response reducible to a congestion game having the same set of players and identical delay functions $f(x) = x$.*

Since the dynamics satisfies the (T, β)-Fairness Condition, we can decompose it into k β-bounded T-coverings $R_1, \dots, R_k$.

Consider a generic β-bounded T-covering $R = \left(S^0, \dots, S^T\right)$. In the following we will often consider the *immediate* costs (or delays) of players during R, that is the cost $c_{\pi(t)}(S^t)$ right after the best response of player $\pi(t)$, for $t = 1, \dots, T$.

Given an optimal strategy profile S^*, since the t-th player $\pi(t)$ performing a best response, before doing it, can always select the strategy she would use in S^*, her immediate cost can be suitably upper bounded as $\sum_{e \in s^*_{\pi(t)}} \left(n_e(S^{t-1}) + 1\right)$.

By extending and strengthening the technique of [10,11], we are able to prove that the best response dynamics satisfying the (T, β)-Fairness Condition fast converges to states approximating the social optimum by a factor $O(\beta)$. It is worth noticing that, by exploiting the technique of [10,11], only a much worse bound of $O(\beta^2)$ could be proved. In order to obtain an $O(\beta)$ bound, we need to develop a different and more involved technique, in which also the functions ρ and H, introduced in [10,11], have to be redefined: roughly speaking, they now must take into account only the last move in R of each player, whereas in [10,11] they were accounting for all the moves in R.

We now introduce functions ρ and H, defined over the set of all the possible β-bounded T-coverings:

- Let $\rho(R) = \sum_{i=1}^{n} \sum_{e \in s_i^*} \left(n_e(S^{\text{last}_R(i)-1}) + 1\right)$;
- let $H(R) = \sum_{i=1}^{n} \sum_{e \in s_i^*} n_e(S^0)$.

Notice that $\rho(R)$ is an upper bound to the sum over all the players of the cost that she would experience on her optimal strategy s_i^* just before her last move in R, whereas $H(R)$ represents the sum over all the players of the delay on the moving player's optimal strategy s_i^* in the initial state S^0 of R. Moreover, since players perform best responses, $\sum_{i=1}^{n} c_i(S^{\text{last}_R(i)}) \leq \rho(R)$, i.e. $\rho(R)$ is an upper bound to the sum of the immediate costs over the last moves of every players.

The upper bound proof is structured as follows. Lemma 1 relates the social cost of the final state S^T of a β-bounded T-covering R with $\rho(R)$, by showing that $C(S^T) \leq 2\rho(R)$. Let $\overline{R}$ and R be two consecutive β-bounded T-coverings; by exploiting Lemmata 2 and 3, providing an upper (lower, respectively) bound to $H(R)$ in terms of $\rho(\overline{R})$ ($\rho(R)$, respectively), Lemma 4 proves that $\frac{\rho}{\text{Opt}}$ rapidly decreases between $\overline{R}$ and R, showing that $\frac{\rho(R)}{\text{Opt}} = O\left(\sqrt{\frac{\rho(\overline{R})}{\text{Opt}}}\right)$. In the proof of Theorem 1, after deriving a trivial upper bound equal to $O(n)$ for $\rho(R_1)$, Lemma 4 is applied to all the $k-1$ couples of consecutive β-bounded T-coverings of the considered dynamics satisfying the (T, β)-Fairness Condition.

The following lemmata show that the social cost at the end of any β-bounded T-covering R is at most $2\rho(R)$, and that $\frac{\rho(\cdot)}{\text{Opt}}$ fast decreases between two consecutive β-bounded T-coverings. They can be proved by adapting some proofs in [10,11] so that they still hold with the new definition of ρ.

Lemma 1. *For any $\beta \geq 1$, given a β-bounded T-covering R, $C(S^T) \leq 2\rho(R)$.*

Lemma 2. *For any $\beta \geq 1$, given a β-bounded T-covering R ending in S^T, $\frac{\sum_{e \in E} n_e(S^T) n_e(S^*)}{\text{Opt}} \leq \sqrt{2\frac{\rho(R)}{\text{Opt}}}$.*

In Lemma 3 we are able to relate $\rho(R)$ and $H(R)$ by much strengthening the technique exploited in [10,11].

Lemma 3. *For any $\beta \geq 1$, given a β-bounded T-covering R, $\frac{\rho(R)}{\text{Opt}} \leq 2\frac{H(R)}{\text{Opt}} + 4\beta + 1$.*

Proof. Let $\bar{N}$ be the set of players changing their strategies by performing best responses in R. First of all, notice that if the players in $\bar{N}$ never select strategies used by some player in S^*, i.e. if they select only resources e such that $n_e(S^*) = 0$, then, by recalling the definitions of $\rho(R)$ and $H(R)$, $\rho(R) \leq H(R) + \text{Opt}$ and the claim would easily follow for any $\beta \geq 1$.

In the following our aim is that of dealing with the generic case in which players moving in R can increase the congestions on resources e such that $n_e(S^*) > 0$.

For every resource $e \in E$, we focus on the congestion on such a resource above a "virtual" congestion frontier $g_e = 2\beta n_e(S^*)$.

We assume that at the beginning of covering R each resource $e \in E$ has a congestion equal to $\delta_{0,e} = \max\{n_e(S^0), g_e\}$, and we call $\delta_{0,e}$ the *congestion of level* 0 on resource e; moreover, $\Delta_0 = \sum_{e \in E} \delta_{0,e} \cdot n_e(S^*)$ is an upper bound to $H(R)$. We refer to Δ_0 as the total congestion of level 0.

The idea is that the total congestion of level 0 can induce on the resources a congestion (over the frontier g_e) being the total congestion of level 1, such a congestion a total congestion of level 2, and so on. More formally, for any $p \geq 1$ and any $e \in E$, we define $\delta_{p,e}$ as the congestion of level p on resource e above the frontier g_e; we say that a congestion $\delta_{p,e}$ of level p on resource e is *induced* by an amount $x_{p,e}$ of congestion of level $p-1$ if some players (say, players in $N_{p,e}$) moving on e can cause such a congestion of level p on e because they are experimenting a delay on the resources of their optimal strategies due to an amount $x_{p,e}$ of congestion of level p. Notice that, for each move of the players in $N_{p,e}$, such an amount $x_{p,e}$ of congestion of level $p-1$ can be used only once, i.e. it cannot be used in order to induce a congestion of level p for other resources in $E \setminus \{e\}$. In other words, $x_{p,e}$ is the overall congestion of level $p-1$ on the resources in the optimal strategies of players in $N_{p,e}$ used in order to induce the congestion $\delta_{p,e}$ of level p on resource e.

For any p, the total congestion of level p is defined as $\Delta_p = \sum_{e \in E} \delta_{p,e} \cdot n_e(S^*)$. Moreover, for any $p \geq 1$, we have that $\sum_{e \in E} x_{p,e} \leq \beta \Delta_{p-1}$ because each player can move at most β times in R and therefore the total congestion of level $p-1$ can be used at most β times in order to induce the total congestion of level p.

It is worth noticing that $\rho(R) \leq \sum_{p=0}^{\infty} \Delta_p + \text{OPT}$, because $\sum_{p=0}^{\infty} \delta_{p,e}$ is an upper bound on the congestion of resource e during the whole covering R:

$$\begin{aligned}
\rho(R) &= \sum_{i=1}^{n} \sum_{e \in s_i^*} \left(n_e(S^{\text{last}(i)-1}) + 1 \right) \\
&\leq \sum_{i=1}^{n} \sum_{e \in s_i^*} \left(\sum_{p=0}^{\infty} \delta_{p,e} + 1 \right) = \sum_{e \in E} \left(n_e(S^*) \left(\sum_{p=0}^{\infty} \delta_{p,e} + 1 \right) \right) \\
&= \sum_{e \in E} \sum_{p=0}^{\infty} \delta_{p,e} n_e(S^*) + \sum_{e \in E} n_e(S^*) = \sum_{p=0}^{\infty} \Delta_p + \text{OPT}
\end{aligned}$$

In the following, we bound $\sum_{p=0}^{\infty} \Delta_p$ from above.

$$\Delta_p = \sum_{e \in E} \delta_{p,e} \cdot n_e(S^*) \leq \sum_{e \in E} \frac{x_{p,e}}{g_e} \cdot n_e(S^*) \leq \sum_{e \in E} \frac{x_{p,e}}{2\beta n_e(S^*)} \cdot n_e(S^*) \leq \frac{\Delta_{p-1}}{2},$$

where the first inequality holds because $\delta_{p,e}$ is the portion of congestion on resource e above the frontier g_e due to some moving players having on the resources of their optimal strategy a delay equal to $x_{p,e}$, and the last inequality holds because each player can move at most β times in R and therefore the total congestion of level $p-1$ can be used at most β times in order to induce the total congestion of level p.

We thus obtain that, for any $p \geq 0$, $\Delta_p \leq \frac{\Delta_0}{2^p}$ and $\sum_{p=0}^{\infty} \Delta_p \leq 2\Delta_0$.

Since $\Delta_0 = \sum_{e \in E} \max\{n_e(S^0), 2\beta n_e(S^*)\} \cdot n_e(S^*) \leq H(R) + 2\beta\textsc{Opt}$ and $\rho(R) \leq \sum_{p=0}^{\infty} \Delta_p + \textsc{Opt} \leq 2\Delta_0 + \textsc{Opt}$, we finally obtain the claim. □

By combining Lemmata 2 and 3, it is possible to prove the following lemma showing that $\frac{\rho(R)}{\textsc{Opt}}$ fast decreases between two consecutive coverings.

Lemma 4. *For any $\beta \geq 1$, given two consecutive β-bounded T-coverings $\overline{R}$ and R, $\frac{\rho(R)}{\textsc{Opt}} \leq 2\sqrt{2\frac{\rho(\overline{R})}{\textsc{Opt}}} + 4\beta + 1$.*

By applying Lemma 4 to all the couples of consecutive β-bounded T-coverings, we are now able to prove the following theorem.

Theorem 1. *Given a linear congestion game, any best response dynamics satisfying the (T, β)-Fairness Condition converges from any initial state to a state S such that $\frac{C(S)}{\textsc{Opt}} = O(\beta)$ in at most $T\lceil \log\log n \rceil$ best responses.*

3.2 Lower Bound

Theorem 2. *For any $\epsilon > 0$, there exist a linear congestion game $\mathcal{G}$ and an initial state S^0 such that, for any $\beta = O(n^{-\frac{1}{\log_2 \epsilon}})$, there exists a best response dynamics starting from S^0 and satisfying the (T, β)-Fairness Condition such that for a number of best responses exponential in n the cost of the reached states is always $\Omega(\beta^{1-\epsilon} \cdot \textsc{Opt})$.*

By choosing $\beta = \sqrt{n}$ and considering a simplified version of the proof giving the above lower bound, it is possible to prove the following corollary. In particular, it shows that even in the case of best response dynamics verifying an $O(n)$-Minimum Liveness Condition, the speed of convergence to efficient states is very slow; such a fact implies that the T-Minimum Liveness condition cannot precisely characterize the speed of convergence to efficient states because it does not capture the notion of fairness in best response dynamics.

Corollary 1. *There exist a linear congestion game $\mathcal{G}$, an initial state S^0 and a best response dynamics starting from S^0 and satisfying the $O(n)$-Minimum Liveness Condition such that for a number of best responses exponential in n the cost of the reached states is always $\Omega\left(\frac{\sqrt[4]{n}}{\log n} \cdot \textsc{Opt}\right)$.*

4 Symmetric Congestion Games

In this section we show that in the symmetric case the unfairness in best response dynamics does not affect the speed of convergence to efficient states. In particular, we are able to show that, for any β, after $T\lceil \log\log n \rceil$ best responses an efficient state is always reached. To this aim, in the following we consider best response dynamics satisfying only the T-Minimum Liveness Condition, i.e. best response dynamics decomposable into k T-coverings $R_1, \ldots, R_k$.

All the results hold for linear congestion games having delay functions $f_e(x) = a_e x + b_e$ with $a_e, b_e \geq 0$ for every $e \in E$. Analogously to the asymmetric case, since our bounds are given as a function of the number of players, the following proposition allows us to focus on congestion games with identical delay functions $f(x) = x$.

Proposition 2. *Any symmetric linear congestion game is best-response reducible to a symmetric congestion game having the same set of players and identical delay functions $f(x) = x$.*

Consider a generic T-covering $R = \left(S^0, \ldots, S^T\right)$. Given an optimal strategy profile S^*, since player i, before performing her last best response, can always select any strategy s_j^*, for $j = 1 \ldots n$, of S^*, her immediate cost $c_i(S^{last(i)})$ can be upper bounded as $\frac{1}{n}\sum_{j=1}^{n}\sum_{e\in s_j^*}(n_e(S^{last(i)-1})+1) = \frac{1}{n}\sum_{e\in E} n_e(S^*)(n_e(S^{last(i)-1})+1)$. In order to prove our upper bound result, we introduce the following function:

- $\Gamma(R) = \frac{1}{n}\sum_{i=1}^{n}\sum_{e\in E} n_e(S^*)(n_e(S^{\mathrm{last}(i)-1}) + 1)$.

Notice that $\Gamma(R)$ is an upper bound to the sum of the immediate cost over the last moves of every players, i.e., $\Gamma(R) \geq \sum_{i=1}^{n} c_i(S^{\mathrm{last}_R(i)})$. Therefore, by exploiting the same arguments used in the proof of Lemma 1, it is possible to prove the following lemma relating the social cost $C(S^T)$ at the end of R with $\Gamma(R)$.

Lemma 5. *Given any T-covering R, $C(S^T) \leq 2\Gamma(R)$.*

Moreover, given any T-covering R, we can relate the social cost $C(S^T)$ of the final state of R with the cost $C(S^0)$ of its initial state.

Lemma 6. *Given any T-covering R, $\frac{C(S^T)}{\textsc{Opt}} \leq (2+2\sqrt{2})\sqrt{\frac{C(S^0)}{\textsc{Opt}}}$.*

Proof.

$$\begin{aligned}
\frac{C(S^T)}{\textsc{Opt}} &\leq \frac{2\Gamma(R)}{\textsc{Opt}} \qquad (1)\\
&= \frac{2}{n\textsc{Opt}}\sum_{i=1}^{n}\sum_{e\in E} n_e(S^*)(n_e(S^{\mathrm{last}(i)-1})+1)\\
&= \frac{2}{n\textsc{Opt}}\left(\sum_{i=1}^{n}\sum_{e\in E} n_e(S^*)n_e(S^{\mathrm{last}(i)-1}) + \sum_{i=1}^{n}\sum_{e\in E} n_e(S^*)\right)\\
&\leq 2 + \frac{2}{n\textsc{Opt}}\sum_{i=1}^{n}\sum_{e\in E} n_e(S^*)n_e(S^{\mathrm{last}(i)-1})\\
&\leq 2 + \frac{2}{n\textsc{Opt}}\sum_{i=1}^{n}\sqrt{\sum_{e\in E} n_e^2(S^*)}\sqrt{\sum_{e\in E} n_e^2(S^{\mathrm{last}(i)-1})} \qquad (2)\\
&= 2 + \frac{2}{n\textsc{Opt}}\sum_{i=1}^{n}\sqrt{\textsc{Opt}}\sqrt{C(S^{\mathrm{last}(i)-1})}
\end{aligned}$$

$$
\begin{aligned}
&\leq 2 + \frac{2}{n\sqrt{\textsc{Opt}}}\sum_{i=1}^{n}\sqrt{2\Phi(S^{\text{last}(i)-1})} && (3)\\
&\leq 2 + \frac{2}{n\sqrt{\textsc{Opt}}}\sum_{i=1}^{n}\sqrt{2\Phi(S^{0})} && (4)\\
&\leq 2 + 2\sqrt{2}\sqrt{\frac{C(S^0)}{\textsc{Opt}}} && (5)\\
&\leq (2 + 2\sqrt{2})\sqrt{\frac{C(S^0)}{\textsc{Opt}}},
\end{aligned}
$$

where inequality (1) follows from Lemma 5, inequality (2) is due to the application of the Cauchy-Schwarz inequality, inequality (3) holds because $C(S^{\text{last}(i)-1}) \leq 2\Phi(S^{\text{last}(i)-1})$, inequality (4) holds because the potential function can only decrease at each best response and inequality (5) holds because $\Phi(S^0) \leq C(S^0)$. □

By applying Lemma 6 to all the pairs of consecutive T-coverings, we are now able to prove the following theorem.

Theorem 3. *Given a linear symmetric congestion game, any best response dynamics satisfying the T-Minimum Liveness Condition converges from any initial state to a state S such that $\frac{C(S)}{\textsc{Opt}} = O(1)$ in at most $T\lceil \log\log n\rceil$ best responses.*

5 Extensions and Final Remarks

All the results extend to the setting of weighted congestion games, in which any player $i \in N$ is associated a weight $w_i \geq 1$; notice that it is possible to assume without loss of generality that $w_i \geq 1$ for any $i \in N$ because it is always possible to suitably scale all the weights (and accordingly the coefficients of the latency functions) in order to obtain such a condition. Let $W = \sum_{i=1}^{n} w_i$. We denote by $l_e(S)$ the congestion on resource e in a state S, i.e. $l_e(S) = \sum_{i|e\in S_i} w_i$. The cost of player i in state S is $c_i(S) = w_i \sum_{e\in S_i} f_e(l_e(S))$. The social cost is given by the sum of the players costs: $C(S) = \sum_{i\in N} c_i(S) = \sum_{e\in E} l_e f_e(l_e(S))$. The following theorems hold.

Theorem 4. *Given a linear weighted congestion game, any best response dynamics satisfying the (T,β)-Fairness Condition converges from any initial state to a state S such that $\frac{C(S)}{\textsc{Opt}} = O(\beta)$ in at most $T\lceil \log\log W\rceil$ best responses.*

Theorem 5. *Given a linear weighted symmetric congestion game, any best response dynamics satisfying the T-Minimum Liveness Condition converges from any initial state to a state S such that $\frac{C(S)}{\textsc{Opt}} = O(1)$ in at most $T\lceil \log\log W\rceil$ best responses.*

As a final remark, it is worth to note that our techniques provide a much faster convergence to efficient states with respect to the previous result in the literature.

In particular, in the symmetric setting, Theorem 3 shows that best response dynamics leads to efficient states much faster than how ϵ-Nash dynamics (i.e., sequences of moves reducing the cost of a player by at least a factor of ϵ) leads to ϵ-Nash equilibria [5]. Furthermore, also in the more general asymmetric setting, Theorem 1 shows that the same holds for fair best response dynamics with respect to ϵ-Nash ones [3].

References

1. Ackermann, H., Röglin, H., Vöcking, B.: On the impact of combinatorial structure on congestion games. J. ACM 55(6) (2008)
2. Awerbuch, B., Azar, Y., Epstein, A.: The price of routing unsplittable flow. In: STOC, pp. 57–66. ACM (2005)
3. Awerbuch, B., Azar, Y., Epstein, A., Mirrokni, V.S., Skopalik, A.: Fast convergence to nearly optimal solutions in potential games. In: ACM Conference on Electronic Commerce, pp. 264–273. ACM (2008)
4. Bilò, V., Fanelli, A., Flammini, M., Moscardelli, L.: Performance of one-round walks in linear congestion games. Theory Comput. Syst. 49(1), 24–45 (2011)
5. Chien, S., Sinclair, A.: Convergence to approximate Nash equilibria in congestion games. In: SODA, pp. 169–178. SIAM (2007)
6. Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: STOC, pp. 67–73. ACM (2005)
7. Christodoulou, G., Mirrokni, V.S., Sidiropoulos, A.: Convergence and Approximation in Potential Games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 349–360. Springer, Heidelberg (2006)
8. Yannakakis, M., Johnson, D.S., Papadimitriou, C.H.: How easy is local search? Journal of Computer and System Sciences 37, 79–100 (1988)
9. Fabrikant, A., Papadimitriou, C.H., Talwar, K.: The complexity of pure Nash equilibria. In: STOC, pp. 604–612. ACM (2004)
10. Fanelli, A., Flammini, M., Moscardelli, L.: The Speed of Convergence in Congestion Games under Best-Response Dynamics. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 796–807. Springer, Heidelberg (2008)
11. Fanelli, A., Moscardelli, L.: On best response dynamics in weighted congestion games with polynomial delays. Distributed Computing 24, 245–254 (2011)
12. Koutsoupias, E., Papadimitriou, C.: Worst-Case Equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
13. Nash, J.F.: Equilibrium points in n-person games. Proceedings of the National Academy of Sciences, 36, 48–49 (1950)
14. Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. International Journal of Game Theory 2, 65–67 (1973)
15. Skopalik, A., Vöcking, B.: Inapproximability of pure Nash equilibria. In: STOC, pp. 355–364. ACM (2008)

Fast Balanced Partitioning Is Hard Even on Grids and Trees*

Andreas Emil Feldmann

Institute of Theoretical Computer Science, ETH Zürich, Switzerland
feldmann@inf.ethz.ch

Abstract. Two kinds of approximation algorithms exist for the k-BALANCED PARTITIONING problem: those that are fast but compute unsatisfactory approximation ratios, and those that guarantee high quality ratios but are slow. In this paper we prove that this tradeoff between *runtime* and *solution quality* is unavoidable. For the problem a minimum number of edges in a graph need to be found that, when cut, partition the vertices into k equal-sized sets. We develop a general reduction which identifies some sufficient conditions on the considered graph class in order to prove the hardness of the problem. We focus on two combinatorially simple but very different classes, namely *trees* and *solid grid graphs*. The latter are finite connected subgraphs of the infinite two-dimensional grid without holes. We apply the reduction to show that for solid grid graphs it is NP-hard to approximate the optimum number of cut edges within any satisfactory ratio. We also consider solutions in which the sets may deviate from being equal-sized. Our reduction is applied to grids and trees to prove that no *fully polynomial time* algorithm exists that computes solutions in which the sets are arbitrarily close to equal-sized. This is true even if the number of edges cut is allowed to increase when the limit on the set sizes decreases. These are the first bicriteria inapproximability results for the k-BALANCED PARTITIONING problem.

1 Model and Setting

We consider the k-BALANCED PARTITIONING problem in which the n vertices of a graph need to be partitioned into k sets of size at most $\lceil n/k \rceil$ each. At the same time the *cut size*, which is the number of edges connecting vertices from different sets, needs to be minimised. This problem has many applications including VLSI circuit design [4], image processing [29], route planning [6], and divide-and-conquer algorithms [23]. In our case the motivation (cf. [2, Section 4]) stems from parallel computations for finite element models (FEMs). In these a continuous domain of a physical model is discretised into a mesh of sub-domains (the elements). The mesh induces a graph in which the vertices are the elements and each edge connects neighbouring sub-domains. A vertex then corresponds to a computational task in the physical simulation, during which tasks that

* The author is supported by the Swiss National Science Foundation under grant 200021_125201/1.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 372–382, 2012.

are adjacent in the graph need to exchange data. Since the model is usually very large, the computation is done in parallel. Hence the tasks need to be scheduled on to k machines (which corresponds to a partition of the vertices) so that the loads of the machines (the sizes of the sets in the partition) are balanced. At the same time the interprocessor communication (the cut size) needs to be minimised since this constitutes a bottleneck in parallel computing. In this paper we focus on 2D FEMs. For these the corresponding graph is a planar graph, typically given by a triangulation or a quadrilateral tiling of the plane [10]. We concentrate on the latter and consider so called *solid grid graphs* which correspond to tessellations into squares. A *grid graph* is a finite subgraph of the infinite 2D grid. An interior face of a grid graph is called a *hole* if more than four edges surround it. If a grid graph is connected and does not have any holes, it is called *solid*.

In general it is NP-hard to approximate the cut size of k-BALANCED PARTITIONING within any finite factor [1]. However the corresponding reduction relies on the fact that a general graph may not be connected and thus the optimal cut size can be zero. Since a 2D FEM always induces a connected planar graph, this strong hardness result may not apply. Yet even for trees [14] it is NP-hard to approximate the cut size within n^c, for any constant $c < 1$. The latter result however relies on the fact that the maximum degree of a tree can be arbitrarily large. Typically though, a 2D FEM induces a graph of constant degrees, as for instance in grid graphs. In fact, even though approximating the cut size in constant degree trees is APX-hard [14], there exists an $\mathcal{O}(\log(n/k))$ approximation algorithm [24] for these. This again raises the question of whether efficient approximation algorithms can be found for graphs induced by 2D FEMs. In this paper we give a negative answer to this question. We prove that it is NP-hard to approximate the cut size within n^c for any constant $c < 1/2$ for solid grid graphs. We also show that this is asymptotically tight by providing a corresponding approximation algorithm.

Hence when each set size is required to be at most $\lceil n/k \rceil$ (the *perfectly balanced* case), the achievable approximation factors are not satisfactory. To circumvent this issue, both in theory and practice it has proven beneficial to consider *bicriteria approximations*. Here additionally the sets may deviate from being perfectly balanced. The computed cut size is compared with the optimal perfectly balanced solution. Throughout this paper we denote the approximation ratio on the cut size by α.

For planar graphs the famous Klein-Plotkin-Rao Theorem [20] can be combined with spreading metric techniques [11] in order to compute a solution for which $\alpha \in \mathcal{O}(1)$ and each set has size at most $2\lceil n/k \rceil$. This needs $\widetilde{\mathcal{O}}(n^3)$ time or $\widetilde{\mathcal{O}}(n^2)$ expected time. For the same guarantee on the set sizes, a faster algorithm exists for solid grid graphs [12]. It runs in $\widetilde{\mathcal{O}}(n^{1.5})$ time but approximates the cut size within $\alpha \in \mathcal{O}(\log k)$. However it is not hard to see how set sizes that deviate by a factor of 2 from being perfectly balanced may be undesirable for practical applications. For instance in parallel computing this means a significant slowdown. This is why graph partitioning heuristics such as Metis [18] or Scotch [5] allow to compute *near-balanced* partitions. Here each set has size at most $(1+\varepsilon)\lceil n/k \rceil$, for arbitrary $\varepsilon > 0$. The heuristics used in practice however do not give any guarantees on the cut size. For general graphs the best algorithm [14] known that gives such a guarantee, will compute a near-balanced solution for which $\alpha \in \mathcal{O}(\log n)$. However the runtime of this algorithm increases exponentially when ε decreases. Therefore this algorithm is too slow for practical purposes. Do algorithms exist that are both *fast* and compute *near-balanced* solutions, and for which rigorous guarantees can be given on the computed cut size? Note that the factor α of the above algorithm does not depend on ε. It therefore suggests itself to devise an algorithm that will compensate the cost of being able to compute near-balanced solutions not in the runtime but in the cut size (as long as it does not increase too much). In this paper however, we show that no such algorithm exists that is reasonable for practical applications. More precisely, we consider *fully polynomial time* algorithms for which the runtime is polynomial in n/ε. We show that, unless P=NP, for solid grid graphs there is no such algorithm for which the computed solution is near-balanced and $\alpha = n^c/\varepsilon^d$, for any constants c and d where $c < 1/2$.

Our main contribution is a general reduction with which hardness results such as the two described above can be generated. For it we identify some sufficient conditions on the considered graphs which make the problem hard. Intuitively these conditions entail that cutting vertices from a graph must be expensive in terms of the number of edges used. We also apply the proposed reduction to general graphs and trees, in order to complement the known results. For general (disconnected) graphs we can show that, unless P=NP, there is no finite value for α allowing a fully polynomial time algorithm that computes near-balanced partitions. For trees we can prove that this is true for any $\alpha = n^c/\varepsilon^d$, for arbitrary constants c and d where $c < 1$. These results demonstrate that the identified sufficient conditions capture a fundamental trait of the k-BALANCED PARTITIONING problem. In particular since we prove the hardness for two combinatorially simple graph classes which however are very dissimilar (as for instance documented by the high *tree-width* of solid grids [8]). For solid grid graphs we harness their isoperimetric properties in order to satisfy the conditions, while for trees we use their ability to have high vertex degrees instead. These are the first bicriteria inapproximability results for the problem. We also show that all of them are asymptotically tight by giving corresponding approximation algorithms.

Related Work. Apart from the results mentioned above, Simon and Teng [28] gave a framework with which bicriteria approximations to k-BALANCED PARTITIONING can be computed. It is a recursive procedure that repeatedly uses a given algorithm for *sparsest cuts*. If a sparsest cut can be approximated within a factor of β then their algorithm yields ratios $\varepsilon = 1$ and $\alpha \in \mathcal{O}(\beta \log k)$. The best factor β for general graphs [3] is $\mathcal{O}(\sqrt{\log n})$. For planar graphs Park and Phillips [26] show how to yield $\beta \in \mathcal{O}(t)$ in $\widetilde{\mathcal{O}}(n^{1.5+1/t})$ time, for arbitrary t. On solid grid graphs constant approximations to sparsest cuts can be computed in linear time [13]. For general graphs the best ratio α is achieved by Krauthgamer *et al.* [21]. For $\varepsilon = 1$ they give an algorithm for which $\alpha \in \mathcal{O}(\sqrt{\log n \log k})$.

Near-balanced partitions were considered by Andreev and Räcke [1] who showed that a ratio of $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$ is possible. This was later improved [14] to $\alpha \in \mathcal{O}(\log n)$, making α independent of ε. In the latter paper also a PTAS is given for trees. For perfectly balanced solutions, there is an approximation algorithm achieving $\alpha \in \mathcal{O}(\Delta \log_\Delta(n/k))$ for trees [24], where Δ is the maximum degree.

The special case when $k = 2$ (the BISECTION problem) has been thoroughly studied. The problem is NP-hard in general [17] and can be approximated within $\mathcal{O}(\log n)$ [27]. Assuming the Unique Games Conjecture, no constant approximations are possible in polynomial time [19]. Leighton and Rao [22] show how near-balanced solutions for which $\alpha \in \mathcal{O}(\beta/\varepsilon^3)$ can be computed, where β is as above. In contrast to the case of arbitrary k, the BISECTION problem can be computed optimally in $\mathcal{O}(n^4)$ time for solid grid graphs [15], and in $\mathcal{O}(n^2)$ time for trees [24]. For planar graphs the complexity of BISECTION is unknown, but a PTAS exists [7].

2 A General Reduction

To derive the hardness results we give a reduction from the 3-PARTITION problem defined below. It is known that 3-PARTITION is strongly NP-hard [16] which means that it remains so even if all integers are polynomially bounded in the size of the input.

Definition 1 (3-PARTITION). *Given $3k$ integers $a_1, \dots, a_{3k}$ and a threshold s such that $s/4 < a_i < s/2$ for each $i \in \{1, \dots 3k\}$, and $\sum_{i=1}^{3k} a_i = ks$, find a partition of the integers into k triples such that each triple sums up to exactly s.*

We will set up a general reduction from 3-PARTITION to different graph classes. This will be achieved by identifying some structural properties that a graph constructed from a 3-PARTITION instance has to fulfil, in order to show the hardness of the k-BALANCED PARTITIONING problem. We will state a lemma which asserts that if the constructed graph has these properties then an algorithm computing near-balanced partitions and approximating the cut size within some α is able to decide the 3-PARTITION problem. We will see that carefully choosing the involved parameters for each of the given graph classes yields the desired reductions. While describing the structural properties we will exemplify them for general (disconnected) graphs which constitute an easily understandable case. For these graphs it is NP-hard to approximate the cut size within any finite factor [1]. We will show that, unless P=NP, no fully polynomial time algorithm exists for any α when near-balanced solutions are desired.

For any 3-PARTITION instance we construct $3k$ graphs, which we will call *gadgets*, with a number of vertices proportional to the integers a_1 to a_{3k}. In particular, for general graphs each gadget G_i, where $i \in \{1, \dots 3k\}$, is a connected graph on $2a_i$ vertices. This assures that the gadgets can be constructed in polynomial time since 3-PARTITION is strongly NP-hard. In general we will assume that we can construct $3k$ gadgets for the given graph class such that each gadget has pa_i vertices for some p specific for the graph class. These gadgets will then be connected using some number m of edges. The parameters p and m may depend on the values of the given 3-PARTITION instance. For the case of general graphs we chose $p = 2$ and we let $m = 0$, i.e. the gadgets are disconnected. In order to show that the given gadgets can be used in a reduction, we require that an upper bound can be given on the number of vertices that can be cut out using a limited number of edges. More precisely, given any colouring of the vertices of all gadgets into k colours, by a *minority vertex* in a gadget G_i we mean a vertex that has the same colour as less than half of G_i's vertices. Any partition of the vertices of all gadgets into k sets induces a colouring of the vertices into k colours. For approximation ratios α and ε, the property we need is that cutting the graph containing n vertices into k sets using at most αm edges, produces less than $p - \varepsilon n$ minority vertices in total. Clearly ε needs to be sufficiently small so that the graph exists. When considering fully polynomial time algorithms, ε should however also not be too small since otherwise the runtime may not be polynomial. For general graphs we achieve this by choosing $\varepsilon = (2ks)^{-1}$. This means that $p - \varepsilon n = 1$ since $n = \sum_{i=1}^{3k} pa_i = 2ks$. Simultaneously the runtime of a corresponding algorithm is polynomial in the size of the 3-PARTITION instance since 3-PARTITION is strongly NP-hard. Additionally the desired condition is met for this graph class since no gadget can be cut using $\alpha m = 0$ edges. The following definition formalises the needed properties.

Definition 2. *For each instance I of 3-PARTITION with integers a_1 to a_{3k} and threshold s, a* reduction set for k-BALANCED PARTITIONING *contains a graph determined by some given parameters $m \geq 0$, $p \geq 1$, $\varepsilon \geq 0$, and $\alpha \geq 1$ which may depend on I. Such a graph constitutes $3k$ gadgets connected through m edges. Each gadget G_i, where $i \in \{1, \ldots, 3k\}$, has pa_i vertices. Additionally, if a partition of the n vertices of the graph into k sets has a cut size of at most αm, then in total there are less than $p - \varepsilon n$ minority vertices in the induced colouring.*

Obviously the involved parameters have to be set to appropriate values in order for the reduction set to exist. For instance p must be an integer and ε must be sufficiently small compared to p and n. Since however the values will vary with the considered graph class we fix them only later. In the following lemma we will assume that the reduction set exists and therefore all parameters were chosen appropriately. It assures that given a reduction set, an approximation algorithm for k-BALANCED PARTITIONING can decide the 3-PARTITION problem. For general graphs we have seen above that a reduction set exists for any finite α and $\varepsilon = (2ks)^{-1}$. This means that a fully polynomial time algorithm for k-BALANCED PARTITIONING computing near-balanced partitions and approximating the cut size within α, can decide the 3-PARTITION problem in polynomial time. Such an algorithm can however not exist, unless P=NP.

Lemma 3. *Let an algorithm $\mathcal{A}$ be given that for any graph in a reduction set for k-BALANCED PARTITIONING computes a partition of the n vertices into k sets of size at most $(1+\varepsilon)\lceil n/k \rceil$ each. If the cut size of the computed solution deviates* *by at most α from the optimal cut size of a perfectly balanced solution, then the algorithm can decide the* `3-PARTITION` *problem.*

Proof. Let k be the value given by a `3-PARTITION` instance I, and let G be the graph corresponding to I in the reduction set. Assume that I has a solution. Then obviously cutting the m edges connecting the gadgets of G gives a perfectly balanced solution to I. Hence in this case the optimal solution has cut size at most m. Accordingly algorithm $\mathcal{A}$ will cut at most αm edges since it approximates the cut size by a factor of α. We will show that in the other case when I does not have a solution, the algorithm cuts more than αm edges. Hence $\mathcal{A}$ can decide the `3-PARTITION` problem and thus the lemma follows.

For the sake of deriving a contradiction, assume that algorithm $\mathcal{A}$ cuts at most αm edges in case the `3-PARTITION` instance I does not permit a solution. Since the corresponding graph G is from a reduction set for k-`BALANCED PARTITIONING`, by Definition 2 this means that from its n vertices, in total less than $p - \varepsilon n$ are minority vertices in the colouring induced by the computed solution of $\mathcal{A}$. Each gadget G_i, where $i \in \{1, \dots 3k\}$, of G has a *majority colour*, i.e. a colour that more than half the vertices in G_i share. This is because the size of G_i is pa_i and we can safely assume that $a_i \geq 2$ (otherwise the instance I is trivial due to $s/4 < a_i < s/2$). The majority colours of the gadgets induce a partition $\mathcal{P}$ of the integers a_i of I into k sets. That is, we introduce a set in $\mathcal{P}$ for each colour and put an integer a_i in a set if the majority colour of G_i equals the colour of the set.

Since we assume that I does not admit a solution, if every set in $\mathcal{P}$ contains exactly three integers there must be some set for which the contained integers do not sum up to exactly the threshold s. On the other hand the bounds on the integers, assuring that $s/4 < a_i < s/2$ for each $i \in \{1, \dots, 3k\}$, mean that in case not every set in $\mathcal{P}$ contains exactly three elements, there must also exist a set for which the contained numbers do not sum up to s. By the pigeonhole principle and the fact that the sum over all a_i equals ks, there must thus be some set T among the k in $\mathcal{P}$ for which the sum of the integers is strictly less than s. Since the involved numbers are integers we can conclude that the sum of the integers in T is in fact at most $s - 1$. Therefore the number of vertices in the gadgets corresponding to the integers in T is at most $p(s-1)$. Let w.l.o.g. the colour of T be 1. Apart from the vertices in these gadgets having majority colour 1, all vertices in G that also have colour 1 must be minority vertices. Hence there must be less than $p(s-1) + p - \varepsilon n$ many vertices with colour 1. Since $\sum_{i=1}^{3k} a_i = ks$ and thus $ps = n/k$, these are less than $n/k - \varepsilon n$.

At the same time the algorithm computes a solution inducing a colouring in which each colour has at most $(1+\varepsilon)n/k$ vertices, since $n = pks$ is divisible by k. This means we can give a lower bound of $n - (k-1)(1+\varepsilon)n/k$ on the number of vertices of a colour by assuming that all other colours have the maximum number of vertices. Since this lower bound equals $(1+\varepsilon)n/k - \varepsilon n$, for any $\varepsilon \geq 0$ we get a contradiction on the upper bound derived above for colour 1. Thus the assumption that the algorithm cuts less than αm edges if I does not have a solution is wrong. □

3 Consequences for Grids and Trees

We will now consider solid grid graphs and trees to show the hardness of the k-BALANCED PARTITIONING problem when restricted to these. For grids we establish our results by considering a set of *rectangular* grid graphs which are connected in a row (Figure 1). By a rectangular grid graph we mean a solid grid graph with the following property. In its natural planar embedding for which the vertices are coordinates in $\mathbb{N}^2$ and the edges have unit length, the straight line edges touching the exterior face form an orthogonal rectangle. The *width* of a rectangular grid graph is the number of vertices sharing the same y-coordinate in this embedding. Accordingly the *height* is the number sharing the same x-coordinate. We first prove that such topologies can be used for reduction sets. We satisfy the conditions by observing that a grid graph resembles a discretised polygon and hence shares their isoperimetric properties. This fact was already used in [25] and we harness these results in the following lemma.

Lemma 4. *Let $\varepsilon \geq 0$ and $\alpha \geq 1$. For any 3-PARTITION instance, let a solid grid graph G be given that consists of $3k$ rectangular grids which are connected in a row using their lower left and lower right vertices by $m = 3k - 1$ edges. Moreover let the height and width of a rectangular grid G_i, where $i \in \{1, \ldots 3k\}$, be $\lceil \sqrt{(3k\alpha)^2 + \varepsilon n}\, \rceil$ and $\lceil \sqrt{(3k\alpha)^2 + \varepsilon n}\, \rceil a_i$, respectively. If ε and α are values for which these grids exist, they form a reduction set for k-BALANCED PARTITIONING.*

Proof. Consider one of the described graphs G for a 3-PARTITION instance. Since both the height and the width of each rectangular grid G_i is greater than αm, using at most αm edges it is not possible to cut across a gadget G_i, neither in horizontal nor in vertical direction. Due to [25, Lemma 2] it follows that with this limited amount of edges, the maximum number of vertices can be cut out from the gadgets by using a square shaped cut in one corner of a single gadget. Such a cut will cut out at most $(\alpha m/2)^2$ vertices. Hence if the vertices of the grid graph G are cut into k sets using at most αm edges, then the induced colouring contains at most $(\alpha m/2)^2$ minority vertices in total. Since the size of each gadget is its height times its width, the parameter p is greater than $(\alpha m)^2 + \varepsilon n$. Hence the number of minority vertices is less than $p - \varepsilon n$. □

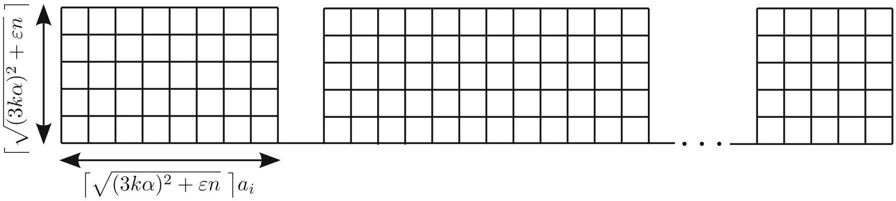

$\lceil \sqrt{(3k\alpha)^2 + \varepsilon n}\, \rceil$
. . .
$\lceil \sqrt{(3k\alpha)^2 + \varepsilon n}\, \rceil a_i$

Fig. 1. The solid grid constructed for the reduction from 3-PARTITION. The gadgets which are rectangular grids are connected through the bottom left and right vertices.

The above topology is first used in the following theorem to show that no satisfying fully polynomial time algorithm exists.

Theorem 5. *Unless P=NP, there is no fully polynomial time algorithm for the k-`BALANCED PARTITIONING` problem on solid grid graphs that for any $\varepsilon > 0$ computes a solution in which each set has size at most $(1+\varepsilon)\lceil n/k \rceil$ and where $\alpha = n^c/\varepsilon^d$, for any constants c and d where $c < 1/2$.*

Proof. In order to prove the claim we need to show that a reduction set as suggested by Lemma 4 exists and can be constructed in polynomial time. We first prove the existence by showing that the number of vertices of a grid graph as suggested by Lemma 4 is finite and hence its construction is feasible. Since $\alpha \geq 1$ we obtain $\lceil \sqrt{(3k\alpha)^2 + \varepsilon n}\, \rceil \leq 2\sqrt{(3k\alpha)^2 + \varepsilon n}$. Thus the parameter p, which is determined by the width and height of the gadgets, is at most $4(3k\alpha)^2 + 4\varepsilon n$. Since the algorithm can compute a near-balanced partition for any $\varepsilon > 0$ we set $\varepsilon = (8ks)^{-1}$. The number of vertices in a solid grid graph as suggested by Lemma 4 is

$$n = \sum_{i=1}^{3k} pa_i \leq (4(3kn^c/\varepsilon^d)^2 + 4\varepsilon n)ks = 36k^3 s(8ks)^{2d} n^{2c} + n/2.$$

Solving this inequality for n gives $n \leq (72k^3 s(8ks)^{2d})^{\frac{1}{1-2c}}$ which is finite since $c < 1/2$. Additionally if c and d are constants, any grid graph in the reduction set can be constructed in polynomial time since `3-PARTITION` is strongly NP-hard. For $\varepsilon = (8ks)^{-1}$ a fully polynomial time algorithm has a runtime that is polynomial in the `3-PARTITION` instance when executed on the corresponding grid. However, unless P=NP, this algorithm cannot exist since it decides the `3-PARTITION` problem due to Lemma 3. □

Next we consider computing perfectly balanced partitions. The proof of the following theorem is deferred to the full version of the paper.

Theorem 6. *There is no polynomial time algorithm for the k-`BALANCED PARTITIONING` problem on solid grid graphs that approximates the cut size within $\alpha = n^c$ for any constant $c < 1/2$, unless P=NP.*

Lemma 4 shows that for solid grid graphs the hardness derives from their isoperimetric properties. Trees do not experience such qualities. However they may have high vertex degrees, which grids cannot. The following theorem shows that this property also leads to a similar hardness as for solid grid graphs. The proof is again deferred to the full version.

Theorem 7. *Unless P=NP, there is no fully polynomial time algorithm for the k-`BALANCED PARTITIONING` problem on trees that for any $\varepsilon > 0$ computes a solution in which each set has size at most $(1+\varepsilon)\lceil n/k \rceil$ and where $\alpha = n^c/\varepsilon^d$, for any constants c and d where $c < 1$.*

4 Conclusions

Are there algorithms for the k-BALANCED PARTITIONING problem that are both *fast* and compute *near-balanced solutions*, even when allowing the cut size to increase when ε decreases? We gave a negative answer to this question. This means that the tradeoff between fast runtime and good solution quality, as provided by the algorithms mentioned in the introduction, is unavoidable. In particular the runtime to compute near-balanced solutions [14] has to increase exponentially with ε. Also our results draw a frontier of the hardness of the problem by showing how the cut size needs to increase with decreasing ε for fully polynomial time algorithms. This gave the first bicriteria inapproximability results for the k-BALANCED PARTITIONING problem.

To show the tightness of the achieved results, for grid graphs we harness results by Diks *et al.* [9]. They provide a polynomial time algorithm to cut out any number of vertices using at most $\mathcal{O}(\sqrt{\Delta n})$ edges from a planar graph with maximum degree Δ. Since $\Delta = 4$ in a grid graph, it is possible to repeatedly cut out $\lceil n/k \rceil$ vertices from the grid using $\mathcal{O}(k\sqrt{n})$ edges in total. A perfectly balanced partition needs at least $k - 1$ edges in a connected graph. Hence we obtain an $\alpha \in \mathcal{O}(\sqrt{n})$ approximation algorithm for grids. This shows that both the hardness results we gave for these graphs are asymptotically tight, since the algorithm runs in (fully) polynomial time. For trees a trivial approximation algorithm can cut all edges in the graph and thereby yield $\alpha = n$. This shows that also the achieved result for trees is asymptotically tight.

We were able to show that both trees and grids experience similar hardness. This is remarkable since these graphs have entirely different combinatorial properties. On the other hand, it emphasizes the ability of the given reduction framework to capture a fundamental trait of the k-BALANCED PARTITIONING problem. It remains to be seen what other structural properties can be harnessed for our framework, in order to prove the hardness for entirely different graph classes. Another interesting approach would be to take the opposite view and identify properties of graphs that in fact lead to good approximation algorithms.

Note that the respective ratios α of the bicriteria inapproximability results can in each case be amplified arbitrarily due to the unrestricted constant d. Also, grids model the graphs resulting from 2D FEMs. For these reasons our results imply that completely different methods (possibly randomness or fixed parameter tractability) must be employed in order to find fast practical algorithms with rigorously bounded approximation guarantees.

References

1. Andreev, K., Räcke, H.: Balanced graph partitioning. Theory of Computing Systems 39(6), 929–939 (2006)
2. Arbenz, P., van Lenthe, G., Mennel, U., Müller, R., Sala, M.: Multi-level μ-finite element analysis for human bone structures. In: Proceedings of the 8th Workshop on State-of-the-art in Scientific and Parallel Computing (PARA), pp. 240–250 (2007)

3. Arora, S., Rao, S., Vazirani, U.: Expander flows, geometric embeddings and graph partitioning. In: Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC), pp. 222–231 (2004)
4. Bhatt, S., Leighton, F.T.: A framework for solving VLSI graph layout problems. Journal of Computer and System Sciences 28(2), 300–343 (1984)
5. Chevalier, C., Pellegrini, F.: PT-Scotch: A tool for efficient parallel graph ordering. Parallel Computing 34(68), 318–331 (2008)
6. Delling, D., Goldberg, A., Pajor, T., Werneck, R.: Customizable route planning. Experimental Algorithms, 376–387 (2011)
7. Díaz, J., Serna, M.J., Torán, J.: Parallel approximation schemes for problems on planar graphs. Acta Informatica 33(4), 387–408 (1996)
8. Diestel, R., Jensen, T.R., Gorbunov, K.Y., Thomassen, C.: Highly connected sets and the excluded grid theorem. Journal of Combinatorial Theory, Series B 75(1), 61–73 (1999)
9. Diks, K., Djidjev, H.N., Sykora, O., Vrto, I.: Edge separators of planar and outerplanar graphs with applications. Journal of Algorithms 14(2), 258–279 (1993)
10. Elman, H., Silvester, D., Wathen, A.: Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics. Oxford University Press, USA (2005)
11. Even, G., Naor, J., Rao, S., Schieber, B.: Fast approximate graph partitioning algorithms. SIAM Journal on Computing 28(6), 2187–2214 (1999)
12. Feldmann, A.E.: Balanced Partitioning of Grids and Related Graphs: A Theoretical Study of Data Distribution in Parallel Finite Element Model Simulations. PhD thesis, ETH Zurich, Diss.-Nr. ETH: 20371 (April 2012)
13. Feldmann, A.E., Das, S., Widmayer, P.: Restricted cuts for bisections in solid grids: A proof via polygons. In: Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), pp. 143–154 (2011)
14. Feldmann, A.E., Foschini, L.: Balanced partitions of trees and applications. In: 29th International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 100–111 (2012)
15. Feldmann, A.E., Widmayer, P.: An $O(n^4)$ time algorithm to compute the bisection width of solid grid graphs. In: Proceedings of the 19th Annual European Symposium on Algorithms (ESA), pp. 143–154 (2011)
16. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co. (1979)
17. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete graph problems. Theoretical Computer Science 1(3), 237–267 (1976)
18. Karypis, G., Kumar, V.: METIS-unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, University of Minnesota (1995)
19. Khot, S.A., Vishnoi, N.K.: The Unique Games Conjecture, integrality gap for cut problems and embeddability of negative type metrics into ℓ_1. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 53–62 (2005)
20. Klein, P., Plotkin, S., Rao, S.: Excluded minors, network decomposition, and multicommodity flow. In: Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC), pp. 682–690 (1993)
21. Krauthgamer, R., Naor, J., Schwartz, R.: Partitioning graphs into balanced components. In: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 942–949 (2009)
22. Leighton, T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. Journal of the ACM 46(6), 787–832 (1999)

23. Lipton, R., Tarjan, R.: Applications of a planar separator theorem. SIAM Journal on Computing 9, 615–627 (1980)
24. MacGregor, R.M.: On Partitioning a Graph: a Theoretical and Empirical Study. PhD thesis, University of California, Berkeley (1978)
25. Papadimitriou, C., Sideri, M.: The bisection width of grid graphs. Theory of Computing Systems 29, 97–110 (1996)
26. Park, J.K., Phillips, C.A.: Finding minimum-quotient cuts in planar graphs. In: Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC), pp. 766–775 (1993)
27. Räcke, H.: Optimal hierarchical decompositions for congestion minimization in networks. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC) (2008)
28. Simon, H.D., Teng, S.H.: How good is recursive bisection? SIAM Journal on Scientific Computing 18(5), 1436–1445 (1997)
29. Wu, Z., Leahy, R.: An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 15(11), 1101–1113 (1993)

A Characterization of Bispecial Sturmian Words

Gabriele Fici

I3S, CNRS & Université Nice Sophia Antipolis, France
fici@i3s.unice.fr

Abstract. A finite Sturmian word w over the alphabet $\{a, b\}$ is left special (resp. right special) if aw and bw (resp. wa and wb) are both Sturmian words. A bispecial Sturmian word is a Sturmian word that is both left and right special. We show as a main result that bispecial Sturmian words are exactly the maximal internal factors of Christoffel words, that are words coding the digital approximations of segments in the Euclidean plane. This result is an extension of the known relation between central words and primitive Christoffel words. Our characterization allows us to give an enumerative formula for bispecial Sturmian words. We also investigate the minimal forbidden words for the set of Sturmian words.

Keywords: Sturmian words, Christoffel words, special factors, minimal forbidden words, enumerative formula.

1 Introduction

Sturmian words are non-periodic infinite words of minimal factor complexity. They are characterized by the property of having exactly $n+1$ distinct factors of length n for every $n \geq 0$ (and therefore are binary words) [16]. As an immediate consequence of this property, one has that in any Sturmian word there is a unique factor for each length n that can be extended to the right with both letters into a factor of length $n + 1$. These factors are called *right special factors*. Moreover, since any Sturmian word is recurrent (every factor appears infinitely often) there is a unique factor for each length n that is left special, i.e., can be extended to the left with both letters into a factor of length $n + 1$.

The set *St* of finite factors of Sturmian words coincides with the set of binary *balanced* words, i.e., binary words having the property that any two factors of the same length have the same number of occurrences of each letter up to one. These words are also called (finite) Sturmian words and have been extensively studied because of their relevant role in several fields of theoretical computer science.

If one considers extendibility within the set *St*, one can define *left special Sturmian words* (resp. *right special Sturmian words*) [9] as those words w over the alphabet $\Sigma = \{a, b\}$ such that aw and bw (resp. wa and wb) are both Sturmian words. For example, the word aab is left special since $aaab$ and $baab$

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 383–394, 2012.

are both Sturmian words, but is not right special since *aabb* is not a Sturmian word.

Left special Sturmian words are precisely the binary words having suffix automaton[1] with minimal state complexity (cf. [11,17]). From combinatorial considerations one has that right special Sturmian words are the reversals of left special Sturmian words.

The Sturmian words that are both left special and right special are called *bispecial Sturmian words*. They are of two kinds: *strictly bispecial Sturmian words*, that are the words w such that awa, awb, bwa and bwb are all Sturmian words, or *non-strictly bispecial Sturmian words* otherwise. Strictly bispecial Sturmian words have been deeply studied (see for example [5,9]) because they play a central role in the theory of Sturmian words. They are also called *central words*. Non-strictly bispecial Sturmian words, instead, received less attention.

One important field in which Sturmian words arise naturally is discrete geometry. Indeed, Sturmian words can be viewed as digital approximations of straight lines in the Euclidean plane. It is known that given a point (p, q) in the discrete plane $\mathbb{Z} \times \mathbb{Z}$, with $p, q > 0$, there exists a unique path that approximates from below (resp. from above) the segment joining the origin $(0, 0)$ to the point (p, q). This path, represented as a concatenation of horizontal and vertical unitary segments, is called the *lower (resp. upper) Christoffel word* associated to the pair (p, q). If one encodes horizontal and vertical unitary segments with the letters a and b respectively, a lower (resp. upper) Christoffel word is always a word of the form awb (resp. bwa), for some $w \in \Sigma^*$. If (and only if) p and q are coprime, the associated Christoffel word is primitive (that is, it is not the power of a shorter word). It is known that a word w is a strictly bispecial Sturmian word if and only if awb is a primitive lower Christoffel word (or, equivalently, if and only if bwa is a primitive upper Christoffel word). As a main result of this paper, we show that this correspondence holds in general between bispecial Sturmian words and Christoffel words. That is, we prove (in Theorem 2) that w is a bispecial Sturmian word if and only if there exist letters x, y in $\{a, b\}$ such that xwy is a Christoffel word.

This characterization allows us to prove an enumerative formula for bispecial Sturmian words (Corollary 1): there are exactly $2n + 2 - \phi(n + 2)$ bispecial Sturmian words of length n, where ϕ is the Euler totient function, i.e., $\phi(n)$ is the number of positive integers smaller than or equal to n and coprime with n. It is worth noticing that enumerative formulae for left special, right special and strictly bispecial Sturmian words were known [9], but to the best of our knowledge we exhibit the first proof of an enumerative formula for non-strictly bispecial (and therefore for bispecial) Sturmian words.

We then investigate minimal forbidden words for the set St of finite Sturmian words. The set of *minimal forbidden words* of a factorial language L is the set of words of minimal length that do not belong to L [15]. Minimal forbidden words represent a powerful tool to investigate the structure of a factorial language

[1] The suffix automaton of a finite word w is the minimal deterministic finite state automaton accepting the language of the suffixes of w.

(see [1]). We give a characterization of minimal forbidden words for the set of Sturmian words in Theorem 3. We show that they are the words of the form ywx such that xwy is a non-primitive Christoffel word, where $\{x, y\} = \{a, b\}$. This characterization allows us to give an enumerative formula for the set of minimal forbidden words (Corollary 2): there are exactly $2(n-1-\phi(n))$ minimal forbidden words of length n for every $n > 1$.

The paper is organized as follows. In Sec. 2 we recall standard definitions on words and factors. In Sec. 3 we deal with Sturmian words and Christoffel words, and present our main result. In Sec. 4 we give an enumerative formula for bispecial Sturmian words. Finally, in Sec. 5, we investigate minimal forbidden words for the language of finite Sturmian words.

2 Words and Special Factors

Let Σ be a finite alphabet, whose elements are called letters. A word over Σ is a finite sequence of letters from Σ. A right-infinite word over Σ is a non-ending sequence of letters from Σ. The set of all words over Σ is denoted by Σ^*. The set of all words over Σ having length n is denoted by Σ^n. The empty word has length zero and is denoted by ε. For a subset X of Σ^*, we note $X(n) = |X \cap \Sigma^n|$. Given a non-empty word w, we let $w[i]$ denote its i-th letter. The reversal of the word $w = w[1]w[2] \cdots w[n]$, with $w[i] \in \Sigma$ for $1 \leq i \leq n$, is the word $\tilde{w} = w[n]w[n-1] \cdots w[1]$. We set $\tilde{\varepsilon} = \varepsilon$. A palindrome is a word w such that $\tilde{w} = w$. A word is called a power if it is the concatenation of copies of another word; otherwise it is called primitive. For a letter $a \in \Sigma$, $|w|_a$ is the number of a's appearing in w. A word w has period p, with $0 < p \leq |w|$, if $w[i] = w[i+p]$ for every $i = 1, \ldots, |w| - p$. Since $|w|$ is always a period of w, every non-empty word has at least one period.

A word z is a factor of a word w if $w = uzv$ for some $u, v \in \Sigma^*$. In the special case $u = \varepsilon$ (resp. $v = \varepsilon$), we call z a prefix (resp. a suffix) of w. We let *Pref*(w), *Suff*(w) and *Fact*(w) denote, respectively, the set of prefixes, suffixes and factors of the word w. The factor complexity of a word w is the integer function $f_w(n) = |Fact(w) \cap \Sigma^n|$, $n \geq 0$.

A factor u of a word w is left special (resp. right special) in w if there exist $a, b \in \Sigma$, $a \neq b$, such that $au, bu \in Fact(w)$ (resp. $ua, ub \in Fact(w)$). A factor u of w is bispecial in w if it is both left and right special. In the case when $\Sigma = \{a, b\}$, a bispecial factor u of w is said to be strictly bispecial in w if aua, aub, bua, bub are all factors of w; otherwise u is said to be non-strictly bispecial in w. For example, let $w = aababba$. The left special factors of w are ε, a, ab, b and ba. The right special factors of w are ε, a, ab and b. Therefore, the bispecial factors of w are ε, a, ab and b. Among these, only ε is strictly bispecial.

In the rest of the paper we fix the alphabet $\Sigma = \{a, b\}$.

3 Sturmian Words and Christoffel Words

A right-infinite word w is called a Sturmian word if $f_w(n) = n+1$ for every $n \geq 0$, that is, if w contains exactly $n+1$ distinct factors of length n for every $n \geq 0$.

Sturmian words are non-periodic infinite words of minimal factor complexity [6]. A famous example of infinite Sturmian word is the Fibonacci word

$$F = abaababaabaababaababaabaababaabaab\cdots$$

obtained as the limit of the substitution $a \mapsto ab$, $b \mapsto a$.

A finite word is called Sturmian if it is a factor of an infinite Sturmian word. Finite Sturmian words are characterized by the following balance property [10]: a finite word w over $\Sigma = \{a, b\}$ is Sturmian if and only if for any $u, v \in Fact(w)$ such that $|u| = |v|$ one has $||u|_a - |v|_a| \leq 1$ (or, equivalently, $||u|_b - |v|_b| \leq 1$). We let St denote the set of finite Sturmian words. The language St is factorial (if $w = uv \in St$, then $u, v \in St$) and extendible (for every $w \in St$ there exist letters $x, y \in \Sigma$ such that $xwy \in St$).

Let w be a finite Sturmian word. The following definitions are in [9].

Definition 1. *A word $w \in \Sigma^*$ is a left special (resp. right special) Sturmian word if $aw, bw \in St$ (resp. if $wa, wb \in St$). A bispecial Sturmian word is a Sturmian word that is both left special and right special. Moreover, a bispecial Sturmian word is strictly bispecial if awa, awb, bwa and bwb are all Sturmian word; otherwise it is non-strictly bispecial.*

We let *LS*, *RS*, *BS*, *SBS* and *NBS* denote, respectively, the sets of left special, right special, bispecial, strictly bispecial and non-strictly bispecial Sturmian words. Hence, $BS = LS \cap RS = SBS \cup NBS$.

The following lemma is a reformulation of a result of de Luca [8].

Lemma 1. *Let w be a word over Σ. Then $w \in LS$ (resp. $w \in RS$) if and only if w is a prefix (resp. a suffix) of a word in SBS.*

Given a bispecial Sturmian word, the simplest criterion to determine if it is strictly or non-strictly bispecial is provided by the following nice characterization [9]:

Proposition 1. *A bispecial Sturmian word is strictly bispecial if and only if it is a palindrome.*

Using the results in [9], one can derive the following classification of Sturmian words with respect to their extendibility.

Proposition 2. *Let w be a Sturmian word. Then:*

- $|\Sigma w \Sigma \cap St| = 4$ *if and only if w is strictly bispecial;*
- $|\Sigma w \Sigma \cap St| = 3$ *if and only if w is non-strictly bispecial;*
- $|\Sigma w \Sigma \cap St| = 2$ *if and only if w is left special or right special but not bispecial;*
- $|\Sigma w \Sigma \cap St| = 1$ *if and only if w is neither left special nor right special.*

Example 1. The word $w = aba$ is a strictly bispecial Sturmian word, since awa, awb, bwa and bwb are all Sturmian words, so that $|\Sigma w \Sigma \cap St| = 4$. The word $w = abab$ is a bispecial Sturmian word since wa, wb, aw and bw are Sturmian words.

Nevertheless, awb is not Sturmian, since it contains aa and bb as factors. So w is a non-strictly bispecial Sturmian word, and $|\Sigma w \Sigma \cap St| = 3$. The Sturmian word $w = aab$ is left special but not right special, and $|\Sigma w \Sigma \cap St| = 2$. Finally, the Sturmian word $w = baab$ is neither left special nor right special, the only word in $\Sigma w \Sigma \cap St$ being awa.

We now recall the definition of central word [9].

Definition 2. *A word over Σ is central if it has two coprime periods p and q and length equal to $p + q - 2$.*

A combinatorial characterization of central words is the following (see [8]):

Proposition 3. *A word w over Σ is central if and only if w is the power of a single letter or there exist palindromes P, Q such that $w = PxyQ = QyxP$, for different letters $x, y \in \Sigma$. Moreover, if $|P| < |Q|$, then Q is the longest palindromic suffix of w.*

Actually, in the statement of Proposition 3, the requirement that the words P and Q are palindromes is not even necessary [5].

We have the following remarkable result [9]:

Proposition 4. *A word over Σ is a strictly bispecial Sturmian word if and only if it is a central word.*

Another class of finite words, strictly related to the previous ones, is that of Christoffel words.

Definition 3. *Let $n > 1$ and $p, q > 0$ be integers such that $p + q = n$. The lower Christoffel word $w_{p,q}$ is the word defined for $1 \leq i \leq n$ by*

$$w_{p,q}[i] = \begin{cases} a \text{ if } iq \ mod(n) > (i-1)q \ mod(n), \\ b \text{ if } iq \ mod(n) < (i-1)q \ mod(n). \end{cases}$$

Example 2. Let $p = 6$ and $q = 4$. We have $\{i4 \bmod(10) \mid i = 0, 1, \ldots, 10\} = \{0, 4, 8, 2, 6, 0, 4, 8, 2, 6, 0\}$. Hence, $w_{6,4} = aababaabab$.

Notice that for every $n > 1$, there are exactly $n - 1$ lower Christoffel words $w_{p,q}$, corresponding to the $n - 1$ pairs (p, q) such that $p, q > 0$ and $p + q = n$.

Remark 1. In the literature, Christoffel words are often defined with the additional requirement that $\gcd(p, q) = 1$ (cf. [3]). We call such Christoffel words primitive, since a Christoffel word is a primitive word if and only if $\gcd(p, q) = 1$.

If one draws a word in the discrete grid $\mathbb{Z} \times \mathbb{Z}$ by encoding each a with a horizontal unitary segment and each b with a vertical unitary segment, the lower Christoffel word $w_{p,q}$ is in fact the best grid approximation from below of the segment joining $(0, 0)$ to (p, q), and has slope q/p, that is, $|w|_a = p$ and $|w|_b = q$ (see Fig. 1).

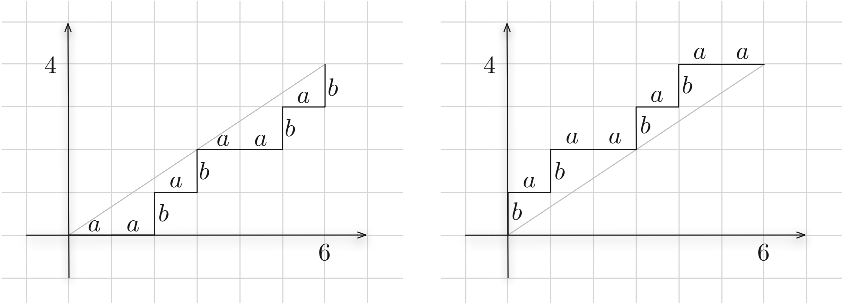


Fig. 1. The lower Christoffel word $w_{6,4} = aababaabab$ (left) and the upper Christoffel word $w'_{6,4} = babaababaa$ (right)

Analogously, one can define the upper Christoffel word $w'_{p,q}$ by

$$w'_{p,q}[i] = \begin{cases} a \text{ if } ip \bmod(n) < (i-1)p \bmod(n), \\ b \text{ if } ip \bmod(n) > (i-1)p \bmod(n). \end{cases}$$

Of course, the upper Christoffel word $w'_{p,q}$ is the best grid approximation from above of the segment joining $(0,0)$ to (p,q) (see Fig. 1).

Example 3. Let $p = 6$ and $q = 4$. We have $\{i6 \bmod(10) \mid i = 0, 1, \ldots, 10\} = \{0, 6, 2, 8, 4, 0, 6, 2, 8, 4, 0\}$. Hence, $w'_{6,4} = babaababaa$.

The next result follows from elementary geometrical considerations.

Lemma 2. *For every pair (p,q) the word $w'_{p,q}$ is the reversal of the word $w_{p,q}$.*

If (and only if) p and q are coprime, the Christoffel word $w_{p,q}$ intersects the segment joining $(0,0)$ to (p,q) only at the end points, and is a primitive word. Moreover, one can prove that $w_{p,q} = aub$ and $w'_{p,q} = bua$ for a palindrome u. Since u is a bispecial Sturmian word and it is a palindrome, u is a strictly bispecial Sturmian word (by Proposition 1). Conversely, given a strictly bispecial Sturmian word u, u is a central word (by Proposition 4), and therefore has two coprime periods p, q and length equal to $p + q - 2$. Indeed, it can be proved that $aub = w_{p,q}$ and $bua = w'_{p,q}$. The previous properties can be summarized in the following theorem (cf. [2]):

Theorem 1. *$SBS = \{w \mid xwy$ is a primitive Christoffel word, $x, y \in \Sigma\}$.*

If instead p and q are not coprime, then there exist coprime integers p', q' such that $p = rp'$, $q = rq'$, for an integer $r > 1$. In this case, we have $w_{p,q} = (w_{p',q'})^r$, that is, $w_{p,q}$ is a power of a primitive Christoffel word. Hence, there exists a central Sturmian word u such that $w_{p,q} = (aub)^r$ and $w'_{p,q} = (bua)^r$. So, we have:

Lemma 3. *The word xwy, $x \neq y \in \Sigma$, is a Christoffel word if and only if $w = (uyx)^n u$, for an integer $n \geq 0$ and a central word u. Moreover, xwy is a primitive Christoffel word if and only if $n = 0$.*

Recall from [8] that the right (resp. left) palindromic closure of a word w is the (unique) shortest palindrome $w^{(+)}$ (resp. $w^{(-)}$) such that w is a prefix of $w^{(+)}$ (resp. a suffix of $w^{(-)}$). If $w = uv$ and v is the longest palindromic suffix of w (resp. u is the longest palindromic prefix of w), then $w^{(+)} = w\tilde{u}$ (resp. $w^{(-)} = \tilde{v}w$).

Lemma 4. *Let xwy be a Christoffel word, $x, y \in \Sigma$. Then $w^{(+)}$ and $w^{(-)}$ are central words.*

Proof. Let xwy be a Christoffel word, $x, y \in \Sigma$. By Lemma 3, $w = (uyx)^n u$, for an integer $n \geq 0$ and a central word u. We prove the claim for the right palindromic closure, the claim for the left palindromic closure will follow by symmetry. If $n = 0$, then $w = u$, so w is a palindrome and then $w^{(+)} = w$ is a central word. So suppose $n > 0$. We first consider the case when u is the power of a single letter (including the case $u = \varepsilon$). We have that either $w = (y^{k+1}x)^n y^k$ or $w = (x^k yx)^n x^k$ for some $k \geq 0$. In the first case, $w^{(+)} = wy = (y^{k+1}x)^n y^{k+1}$, whereas in the second case $w^{(+)} = wyx^k = (x^k yx)^n x^k yx^k$. In both cases one has that $w^{(+)}$ is a strictly bispecial Sturmian word, and thus, by Proposition 4, a central word.

Let now u be not the power of a single letter. Hence, by Proposition 3, there exist palindromes P, Q such that $u = PxyQ = QyxP$. Now, observe that

$$w = (uyx)^n u = Pxy(QyxPxy)^n Q$$

We claim that the longest palindromic suffix of w is $(QyxPxy)^n Q$. Indeed, the longest palindromic suffix of w cannot be w itself since w is not a palindrome, so since any palindromic suffix of w longer than $(QyxPxy)^n Q$ must start in u, in order to prove the claim it is enough to show that the first non-prefix occurrence of u in w is that appearing as a prefix of $(QyxPxy)^n Q$. Now, since the prefix $v = PxyQyxP$ of w can be written as $v = uyxP = Pxyu$, one has by Proposition 3 that v is a central word. It is easy to prove (see, for example, [4]) that the longest palindromic suffix of a central word does not have internal occurrences, that is, appears in the central word only as a prefix and as a suffix. Therefore, since $|u| > |P|$, u is the longest palindromic suffix of v (by Proposition 3), and so appears in v only as a prefix and as a suffix. This shows that $(QyxPxy)^n Q$ is the longest palindromic suffix of w.

Thus, we have $w^{(+)} = wyxP$, and we can write:

$$\begin{aligned} w^{(+)} &= Pxy(QyxPxy)^n QyxP \\ &= PxyQ \cdot yx \cdot P(xyQyxP)^n \\ &= (PxyQyx)^n P \cdot xy \cdot QyxP \end{aligned}$$

so that $w^{(+)} = uyxz = zxyu$ for the palindrome $z = P(xyQyxP)^n = (PxyQyx)^n P$. By Proposition 3, $w^{(+)}$ is a central word. □

We are now ready to state our main result.

Theorem 2. *$BS = \{w \mid xwy$ is a Christoffel word, $x, y \in \Sigma\}$.*

Proof. Let xwy be a Christoffel word, $x, y \in \Sigma$. Then, by Lemma 3, w is of the form $w = (uyx)^n u$, $n \geq 0$, for a central word u. By Lemma 4, w is a prefix of the central word $w^{(+)}$ and a suffix of the central word $w^{(-)}$, and therefore, by Proposition 4 and Lemma 1, w is a bispecial Sturmian word.

Conversely, let w be a bispecial Sturmian word, that is, suppose that the words xw, yw, wx and wy are all Sturmian. If w is strictly bispecial, then w is a central word by Proposition 4, and xwy is a (primitive) Christoffel word by Theorem 1. So suppose $w \in NBS$. By Lemma 3, it is enough to prove that w is of the form $w = (uyx)^n u$, $n \geq 1$, for a central word u and letters $x \neq y$. Since w is not a strictly bispecial Sturmian word, it is not a palindrome (by Proposition 1). Let u be the longest palindromic border of w (that is, the longest palindromic prefix of w that is also a suffix of w), so that $w = uyzxu$, $x \neq y \in \Sigma$, $z \in \Sigma^*$. If $z = \varepsilon$, $w = uyxu$ and we are done. Otherwise, it must be $z = xz'y$ for some $z' \in \Sigma^*$, since otherwise either the word yw would contain yuy and xxu as factors (a contradiction with the hypothesis that yw is a Sturmian word) or the word wx would contain uyy and xux as factors (a contradiction with the hypothesis that wx is a Sturmian word).

So $w = uyxz'yxu$. If $u = \varepsilon$, then it must be $z = (yx)^k$ for some $k \geq 0$, since otherwise either xx would appear as a factor in w, and therefore the word yw would contain xx and yy as factors, being not a Sturmian word, or yy would appear as a factor in w, and therefore the word wx would contain xx and yy as factors, being not a Sturmian word. Hence, if $u = \varepsilon$ we are done, and so we suppose $|u| > 0$.

By contradiction, suppose that w is not of the form $w = (uyx)^n u$. That is, let $w = (uyx)^k u'av$, with $k \geq 1$, $v \in \Sigma^*$, $u'b \in \mathit{Pref}(uyx)$, for different letters a and b. If $|u'| \geq |u|$, then either $|u'| = |u|$ or $|u'| = |u| + 1$. In the first case, $u' = u$ and $w = (uyx)^k uxv'$, for some $v' \in \Sigma^*$, and then the word yw would contain yuy and xux as factors, being not a Sturmian word. In the second case, $u' = uy$ and $w = (uyx)^k uyyv''$, for some $v'' \in \Sigma^*$; since xu is a suffix of w, and therefore $w = (uyx)^k v'''xu$ for some $v''' \in \Sigma^*$, we would have that the word wx contains both uyy and xux as factors, being not a Sturmian word. Thus, we can suppose $u'b \in \mathit{Pref}(u)$. Now, if $a = x$ and $b = y$, then the word yw would contain the factors $yu'y$ and $xu'x$, being not a Sturmian word; if instead $a = y$ and $b = x$, let $u = u'xu''$, so that we can write $w = (uyx)^k u'yv = (uyx)^{k-1} u'xu''yxu'yv$. The word wx would therefore contain the factors $u''yxu'y$ and $xux = xu'xu''x$ (since xu is a suffix of w), being not a Sturmian word (see Fig. 2). In all the cases we obtain a contradiction and the proof is thus complete. □

So, bispecial Sturmian words are the maximal internal factors of Christoffel words. Every bispecial Sturmian word is therefore of the form $w = (uyx)^n u$, $n \geq 0$, for different letters x, y and a central word u. The word w is strictly bispecial if and only if $n = 0$. If $n = 1$, w is a *semicentral word* [4], that is, a

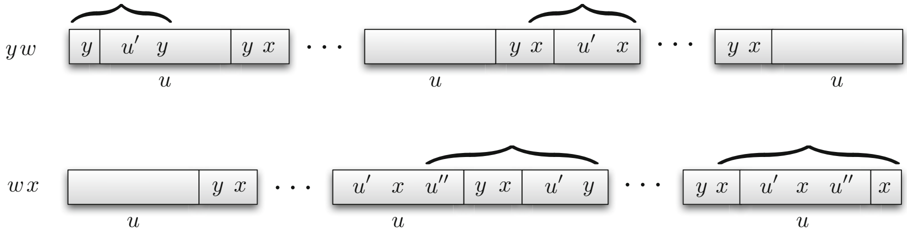


Fig. 2. The proof of Theorem 2

word in which the longest repeated prefix, the longest repeated suffix, the longest left special factor and the longest right special factor all coincide.

4 Enumeration of Bispecial Sturmian Words

In this section we give an enumerative formula for bispecial Sturmian words. It is known that the number of Sturmian words of length n is given by

$$St(n) = 1 + \sum_{i=1}^{n} (n - i + 1)\phi(i)$$

where ϕ is the Euler totient function, i.e., $\phi(n)$ is the number of positive integers smaller than or equal to n and coprime with n (cf. [13,14]).

Let w be a Sturmian word of length n. If w is left special, then aw and bw are Sturmian words of length $n+1$. If instead w if not left special, then only one between aw and bw is a Sturmian word of length $n+1$. Therefore, we have $LS(n) = St(n+1) - St(n)$ and hence

$$LS(n) = \sum_{i=1}^{n+1} \phi(i)$$

Using a symmetric argument, one has that also

$$RS(n) = \sum_{i=1}^{n+1} \phi(i)$$

Since [9] $SBS(n) = LS(n+1) - LS(n) = RS(n+1) - RS(n)$, we have

$$SBS(n) = \phi(n+2)$$

Therefore, in order to find an enumerative formula for bispecial Sturmian words, we only have to enumerate the non-strictly bispecial Sturmian words. We do this in the next proposition.

Proposition 5. *For every $n > 1$, one has*

$$NBS(n) = 2\,(n + 1 - \phi(n + 2))$$

Proof. Let

$$W_n = \{w \mid \ awb \text{ is a lower Christoffel word of length } n + 2\}$$

and

$$W'_n = \{w' \mid \ bw'a \text{ is an upper Christoffel word of length } n + 2\}$$

By Theorem 2, the bispecial Sturmian words of length n are the words in $W_n \cup W'_n$.

Among the $n + 1$ words in W_n, there are $\phi(n + 2)$ strictly bispecial Sturmian words, that are precisely the palindromes in W_n. The $n+1-\phi(n+2)$ words in W_n that are not palindromes are non-strictly bispecial Sturmian words. The other non-strictly bispecial Sturmian words of length n are the $n + 1 - \phi(n + 2)$ words in W'_n that are not palindromes. Since the words in W'_n are the reversals of the words in W_n, and since no non-strictly bispecial Sturmian word is a palindrome by Proposition 1, there are a total of $2(n + 1 - \phi(n + 2))$ non-strictly bispecial Sturmian words of length n. □

Corollary 1. *For every $n \geq 0$, there are $2(n+1) - \phi(n+2)$ bispecial Sturmian words of length n.*

Example 4. The Christoffel words of length 12 and their maximal internal factors, the bispecial Sturmian words of length 10, are reported in Table 1.

5 Minimal Forbidden Words

Given a factorial language L (that is, a language containing all the factors of its words) over the alphabet Σ, a word $v \in \Sigma^*$ is a minimal forbidden word for L if v does not belong to L but every proper factor of v does (see [7] for further details). Minimal forbidden words represent a powerful tool to investigate the structure of a factorial language (cf. [1]). In the next theorem, we give a characterization of the set *MFSt* of minimal forbidden words for the language *St*.

Theorem 3. *$MFSt = \{ywx \mid xwy$ is a non-primitive Christoffel word, $x, y \in \Sigma\}$.*

Proof. If xwy is a non-primitive Christoffel word, then by Theorems 1 and 2, w is a non-strictly bispecial Sturmian word. This implies that ywx is not a Sturmian word, since a word w such that xwy and ywx are both Sturmian is a central word [9], and therefore a strictly bispecial Sturmian word (Proposition 4). Since yw and wx are Sturmian words, we have $ywx \in MFSt$.

Conversely, let $ywx \in MFSt$. By definition, yw is Sturmian, and therefore ywy must be a Sturmian word since *St* is an extendible language. Analogously, since wx is Sturmian, the word xwx must be a Sturmian word. Thus, w is a bispecial Sturmian word, and since $ywx \notin St$, w is a non-strictly bispecial Sturmian word. By Theorems 1 and 2, xwy is a non-primitive Christoffel word. □

Table 1. The Christoffel words of length 12. Their maximal internal factors are the bispecial Sturmian words of length 10. There are $4 = \phi(12)$ strictly bispecial Sturmian words, that are the palindromes *aaaaaaaaaa*, *ababaababa*, *bababbabab* and *bbbbbbbbbb* (underlined), and $14 = 2(11 - 4)$ non-strictly bispecial Sturmian words: *aaaaabaaaa*, *aaaabaaaaa*, *aaabaaabaa*, *aabaaabaaa*, *aabaabaaba*, *abaabaabaa*, *ababababab*, *bababababa*, *babbabbabb*, *bbabbabbab*, *bbabbbabbb*, *bbbabbbabb*, *bbbbabbbbb* and *bbbbbabbbb*.

pair (p, q)	lower Christoffel word $w_{p,q}$	upper Christoffel word $w'_{p,q}$
$(11, 1)$	$a\underline{aaaaaaaaaa}b$	$b\underline{aaaaaaaaaa}a$
$(10, 2)$	$aaaaabaaaaab$	$baaaaabaaaaa$
$(9, 3)$	$aaabaaabaaab$	$baaabaaabaaa$
$(8, 4)$	$aabaabaabaab$	$baabaabaabaa$
$(7, 5)$	$a\underline{ababaababa}b$	$b\underline{ababaababa}a$
$(6, 6)$	$abababababab$	$babababababa$
$(5, 7)$	$a\underline{bababbabab}b$	$b\underline{bababbabab}a$
$(4, 8)$	$abbabbabbabb$	$bbabbabbabba$
$(3, 9)$	$abbbabbbabbb$	$bbbabbbabbba$
$(2, 10)$	$abbbbbabbbbb$	$bbbbbabbbbba$
$(1, 11)$	$a\underline{bbbbbbbbbb}b$	$b\underline{bbbbbbbbbb}a$

Corollary 2. *For every $n > 1$, one has*

$$MFSt(n) = 2(n - 1 - \phi(n))$$

It is known from [14] that $St(n) = O(n^3)$, as a consequence of the estimation (see [12], p. 268)

$$\sum_{i=1}^{n} \phi(i) = \frac{3n^2}{\pi^2} + O(n \log n) \qquad (1)$$

From (1) and from the formula of Corollary 2, we have that

$$\sum_{i=1}^{n} MFSt(n) = O(n^2)$$

References

1. Béal, M.-P., Mignosi, F., Restivo, A.: Minimal Forbidden Words and Symbolic Dynamics. In: Puech, C., Reischuk, R. (eds.) STACS 1996. LNCS, vol. 1046, pp. 555–566. Springer, Heidelberg (1996)
2. Berstel, J., de Luca, A.: Sturmian Words, Lyndon Words and Trees. Theoret. Comput. Sci. 178(1-2), 171–203 (1997)

3. Berstel, J., Lauve, A., Reutenauer, C., Saliola, F.: Combinatorics on Words: Christoffel Words and Repetition in Words. CRM monograph series, vol. 27. American Mathematical Society (2008)
4. Bucci, M., De Luca, A., Fici, G.: Enumeration and structure of trapezoidal words (submitted, 2012)
5. Carpi, A., de Luca, A.: Central Sturmian Words: Recent Developments. In: De Felice, C., Restivo, A. (eds.) DLT 2005. LNCS, vol. 3572, pp. 36–56. Springer, Heidelberg (2005)
6. Coven, E.M., Hedlund, G.A.: Sequences with minimal block growth. Mathematical Systems Theory 7(2), 138–153 (1973)
7. Crochemore, M., Mignosi, F., Restivo, A.: Minimal Forbidden Words and Factor Automata. In: Brim, L., Gruska, J., Zlatuška, J. (eds.) MFCS 1998. LNCS, vol. 1450, pp. 665–673. Springer, Heidelberg (1998)
8. de Luca, A.: Sturmian Words: Structure, Combinatorics, and Their Arithmetics. Theoret. Comput. Sci. 183(1), 45–82 (1997)
9. de Luca, A., Mignosi, F.: Some combinatorial properties of Sturmian words. Theoret. Comput. Sci. 136(2), 361–385 (1994)
10. Dulucq, S., Gouyou-Beauchamps, D.: Sur les Facteurs des Suites de Sturm. Theoret. Comput. Sci. 71(3), 381–400 (1990)
11. Fici, G.: Special Factors and the Combinatorics of Suffix and Factor Automata. Theoret. Comput. Sci. 412(29), 3604–3615 (2011)
12. Hardy, G.H., Wright, E.M.: An Introduction to the Theory of Numbers, 5th edn. Clarendon Press, Oxford (1979)
13. Lipatov, E.P.: A classification of binary collections and properties of homogeneity classes (Russian). Problemy Kibernet. 39, 67–84 (1982)
14. Mignosi, F.: On the number of factors of Sturmian words. Theoret. Comput. Sci. 82, 71–84 (1991)
15. Mignosi, F., Restivo, A., Sciortino, M.: Words and forbidden factors. Theoret. Comput. Sci. 273(1-2), 99–117 (2002)
16. Morse, M., Hedlund, G.A.: Symbolic dynamics II: Sturmian Trajectories. Amer. J. Math. 62, 1–42 (1940)
17. Sciortino, M., Zamboni, L.Q.: Suffix Automata and Standard Sturmian Words. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 382–398. Springer, Heidelberg (2007)

Online Sum-Radii Clustering[⋆]

Dimitris Fotakis[1] and Paraschos Koutris[2]

[1] School of Electrical and Computer Engineering,
National Technical University of Athens, 157 80 Athens, Greece
fotakis@cs.ntua.gr
[2] Computer Science and Engineering, University of Washington, U.S.A.
pkoutris@cs.washington.edu

Abstract. In Online Sum-Radii Clustering, n demand points arrive online and must be irrevocably assigned to a cluster upon arrival. The cost of each cluster is the sum of a fixed opening cost and its radius, and the objective is to minimize the total cost of the clusters opened by the algorithm. We show that the deterministic competitive ratio of Online Sum-Radii Clustering for general metric spaces is $\Theta(\log n)$, where the upper bound follows from a primal-dual online algorithm, and the lower bound is valid for ternary Hierarchically Well-Separated Trees (HSTs) and for the Euclidean plane. Combined with the results of (Csirik et al., MFCS 2010), this result demonstrates that the deterministic competitive ratio of Online Sum-Radii Clustering changes abruptly, from constant to logarithmic, when we move from the line to the plane. We also show that Online Sum-Radii Clustering in HSTs is closely related to the Parking Permit problem introduced by (Meyerson, FOCS 2005). Exploiting the relation to Parking Permit, we obtain a lower bound of $\Omega(\log\log n)$ on the randomized competitive ratio of Online Sum-Radii Clustering in tree metrics. Moreover, we present a simple randomized $O(\log n)$-competitive algorithm, and a deterministic $O(\log\log n)$-competitive algorithm for the fractional version of the problem.

1 Introduction

In clustering problems, we seek a partitioning of n demand points into k groups, or *clusters*, so that a given objective function, that depends on the distance between points in the same cluster, is minimized. Typical examples are the k-Center problem, where we minimize the maximum cluster diameter, the Sum-k-Radii problem, where we minimize the sum of cluster radii, and the k-Median problem, where we minimize the total distance of points to the nearest cluster center. These are fundamental problems in Computer Science, with many important applications, and have been extensively studied from an algorithmic viewpoint (see e.g., [18] and the references therein).

In this work, we study an online clustering problem closely related to Sum-k-Radii. In the online setting, the demand points arrive one-by-one and must be irrevocably assigned to a cluster upon arrival. We require that once formed, clusters cannot be merged,

[⋆] This work was supported by the project AlgoNow, co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning", under the research funding program Thales.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 395–406, 2012.

split, or have their center or radius changed. The goal is to open a few clusters with a small sum of radii. However, instead of requiring that at most k clusters open, which would lead to an unbounded competitive ratio, we follow [6,7] and consider a Facility-Location-like relaxation of Sum-k-Radii, called *Sum-Radii Clustering*. In Sum-Radii Clustering, the cost of each cluster is the sum of a fixed opening cost and its radius, and we seek to minimize the total cost of the clusters opened by the algorithm.

In addition to clustering and data analysis, Sum-Radii Clustering has applications to location problems of wireless base stations, such as sensors [7,8] or antennas [3,15]. In such problems, we place some wireless base stations and setup their communication range so that some communication demands are satisfied and the total setup and operational cost is minimized. A standard assumption is that the setup cost is proportional to the number of stations installed, and the operational cost for each station is proportional to its range (or a low-degree polynomial of it).

Related Work. In the offline setting, Sum-k-Radii and the closely related problem of Sum-k-Diameters[1] have been thoroughly studied. Sum-k-Radii is $\mathbf{NP}$-hard even in metric spaces of constant doubling dimension [14]. Gibson et al. [13] proved that Sum-k-Radii in Euclidean spaces of constant dimension is polynomially solvable, and presented an $O(n^{\log \Delta \, \log n})$-time algorithm for Sum-$k$-Radii in general metric spaces, where Δ is the diameter [14]. As for approximation algorithms, Doddi et al. [9] proved that it is $\mathbf{NP}$-hard to approximate Sum-k-Diameters in general metric spaces within a factor less than 2, and gave a bicriteria algorithm that achieves a logarithmic approximation using $O(k)$ clusters. Subsequently, Charikar and Panigraphy [6] presented a primal-dual $(3.504 + \varepsilon)$-approximation algorithm for Sum-k-Radii in general metric spaces, which uses as a building block a primal-dual 3-approximation algorithm for Sum-Radii Clustering. Biló et al. [3] considered a generalization of Sum-k-Radii, where the cost is the sum of the α-th power of the clusters radii, for $\alpha \geq 1$, and presented a polynomial-time approximation scheme for Euclidean spaces of constant dimension.

Charikar and Panigraphy [6] also considered the incremental version of Sum-k-Radii, Similarly to the online setting, an incremental algorithm processes the demands one-by-one and assigns them to a cluster upon arrival. However, an incremental algorithm can also merge any of its clusters at any time. They presented an $O(1)$-competitive incremental algorithm for Sum-k-Radii that uses $O(k)$ clusters.

In the online setting, where cluster reconfiguration is not allowed, the Unit Covering and the Unit Clustering problems have received considerable attention. In both problems, the demand points arrive one-by-one and must be irrevocably assigned to unit-radius balls upon arrival, so that the number of balls used is minimized. The difference is that in Unit Covering, the center of each ball is fixed when the ball is first used, while in Unit Clustering, there is no fixed center and a ball may shift and cover more demands. Charikar et al. [5] proved an upper bound of $O(2^d d \log d)$ and a lower bound of $\Omega(\log d / \log\log\log d)$ on the deterministic competitive ratio of Unit Covering in d dimensions. The results of [5] imply a competitive ratio of 2 and 4 for Unit Covering on the line and the plane, respectively. The Unit Clustering problem was introduced by Chan and Zarrabi-Zadeh [4]. The deterministic competitive ratio of Unit Clustering on

[1] These problems are closely related in the sense that a c-competitive algorithm for Sum-k-Radii implies a $2c$-competitive algorithm for Sum-k-Diameters, and vice versa.

the line is at most $5/3$ [10] and no less than $8/5$ [11]. Unit Clustering has also been studied in d-dimensions with respect to the L_∞ norm, where the competitive ratio is at most $\frac{5}{6}2^d$, for any d, and no less than $13/6$, for $d \geq 2$ [10].

Departing from this line of work, Csirik at el. [7] studied online clustering to minimize the sum of the setup costs and the diameters of the clusters (CSDF). Motivated by the difference between Unit Covering and Unit Clustering, they considered three models, the strict, the intermediate, and the flexible one, depending on whether the center and the radius of a new cluster are fixed at its opening time. Csirik at el. only studied CSDF on the line and proved that its deterministic competitive ratio is $1 + \sqrt{2}$ for the strict and the intermediate model and $(1 + \sqrt{5})/2$ for the flexible model. Recently, Divéki and Imreh [8] studied online clustering in two dimensions to minimize the sum of the setup costs and the area of the clusters. They proved that the competitive ratio of this problem lies in $(2.22, 9]$ for the strict model and in $(1.56, 7]$ for the flexible model.

Contribution. Following [7], it is natural and interesting to study the online clustering problem of CSDF in metric spaces more general than the line. In this work, we consider the closely related problem of Online Sum-Radii Clustering (OnlSumRad), and give upper and lower bounds on its deterministic and randomized competitive ratio for general metric spaces and for the Euclidean plane. We restrict our attention to the strict model of [7], where the center and the radius of each new cluster are fixed at opening time. To justify our choice, we show that a c-competitive algorithm for the strict model implies an $O(c)$-competitive algorithm for the intermediate and the flexible model.

We show that the deterministic competitive ratio of OnlSumRad for general metric spaces is $\Theta(\log n)$, where the upper bound follows from a primal-dual algorithm, and the lower bound is valid for ternary Hierarchically Well-Separated Trees (HSTs) and for the Euclidean plane. This result is particularly interesting because it demonstrates that the deterministic competitive ratio of OnlSumRad (and of CSDF) changes abruptly, from constant to logarithmic, when we move from the line to the plane. Interestingly, this does not happen when the cost of each cluster is proportional to its area [8].

Another interesting finding is that OnlSumRad in metric spaces induced by HTSs is closely related to the Parking Permit problem introduced by Meyerson [17]. In Parking Permit, we cover a set of driving days by choosing among K permit types, each with a given cost and duration. The permit costs are concave, in the sense that the cost per day decreases with the duration. The algorithm is informed of the driving days in an online fashion, and irrevocably decides on the permits to purchase, so that all driving days are covered by a permit and the total cost is minimized. Meyerson [17] proved that the competitive ratio of Parking Permit is $\Theta(K)$ for deterministic and $\Theta(\log K)$ for randomized algorithms. We prove that OnlSumRad in HSTs is a generalization of Parking Permit. Combined with the randomized lower bound of [17], this implies a lower bound of $\Omega(\log \log n)$ on the randomized competitive ratio of OnlSumRad. Moreover, we show that, under some assumptions, a c-competitive algorithm for Parking Permit with K types implies a c-competitive algorithm for OnlSumRad in HSTs with K levels.

We conclude with a simple and memoryless randomized $O(\log n)$-competitive algorithm, and a deterministic $O(\log \log n)$-competitive algorithm for the fractional version of OnlSumRad. Both algorithms work for general metric spaces. The fractional algorithm is based on the primal-dual approach of [2,1], and generalizes the fractional algorithm of [17] for Parking Permit. We leave as an open problem the existence of a randomized rounding procedure that converts the fractional solution to an (integral) clustering of cost within a constant factor of the original cost. This would imply a randomized $O(\log\log n)$-competitive algorithm for OnlSumRad in general metrics.

Other Related Work. OnlSumRad is a special case of Online Set Cover [2] with sets of different weight. In [2], it is presented a nearly optimal deterministic $O(\log m \log N)$-competitive algorithm, where N is the number of elements and m is the number of sets. Moreover, if all sets have the same weight and each element belongs to at most d sets, the competitive ratio can be improved to $O(\log d \log N)$. If we cast OnlSumRad as a special case of Online Set Cover, N is the number of points in the metric space, which can be much larger than the number of demands n, $m = \Omega(n)$, and $d = O(\log n)$. Hence, a direct application of the algorithm of [2] to OnlSumRad does not lead to an optimal deterministic competitive ratio. This holds even if one could possibly extend the improved ratio of $O(\log d \log N)$ to the weighted set structure of OnlSumRad.

At the conceptual level, OnlSumRad is related to the problem of Online Facility Location [16,12]. However, the two problems exhibit a different behavior w.r.t. their competitive ratio, since the competitive ratio of Online Facility Location is $\Theta(\frac{\log n}{\log\log n})$, even on the line, for both deterministic and randomized algorithms [12].

2 Notation, Problem Definition, and Preliminaries

We consider a metric space (M, d), where M is the set of points and $d : M \times M \mapsto \mathbb{N}$ is the distance function, which is non-negative, symmetric and satisfies the triangle inequality. For a set of points $M' \subseteq M$, we let $\mathrm{diam}(M') \equiv \max_{u,v\in M'}\{d(u,v)\}$ be the diameter and $\mathrm{rad}(M') \equiv \min_{u\in M'} \max_{v\in M'}\{d(u,v)\}$ be the radius of M'. In a *tree metric*, the points correspond to the nodes of an edge-weighted tree and the distances are given by the tree's shortest path metric. For some $\alpha > 1$, a *Hierarchically α-Well-Separated Tree* (α-HST) is a complete rooted tree with lengths on its edges such that: (i) the distance of each leaf to its parent is 1, and (ii) on every path from a leaf to the root, the edge length increases by a factor of α on every level. Thus, the distance of any node v_k at level k to its children is α^{k-1} and the distance of v_k to the nearest leaf is $(\alpha^k - 1)/(\alpha - 1)$. We usually identify a tree with the metric space induced by it.

A *cluster* $C(p, r) \equiv \{v : d(p, v) \le r\}$ is determined by its center p and its radius r, and consists of all points within a distance at most r to p. The cost of a cluster $C(p, r)$ is the sum of its opening cost f and its radius r.

Sum-Radii Clustering. In the offline version of Sum-Radii Clustering, we are given a metric space (M, d), a cluster opening cost f, and a set $D = \{u_1, \dots, u_n\}$ of demand points in M. The goal is to find a collection of clusters $C(p_1, r_1), \dots, C(p_k, r_k)$ that cover all demand points in D and minimize the total cost, which is $\sum_{i=1}^{k}(f + r_i)$.

Online Sum-Radii Clustering. In the online setting, the demand points arrive one-by-one, in an online fashion, and must be irrevocably assigned to an open cluster upon arrival. Formally, the input to Online Sum-Radii Clustering (OnlSumRad) consists of the cluster opening cost f and a sequence $u_1, \dots, u_n$ of (not necessarily distinct) demand points in an underlying metric space (M, d). The goal is to maintain a set of clusters of minimum total cost that cover all demand points revealed so far.

In this work, we focus on the so-called Fixed-Cluster version of OnlSumRad, where the center and the radius of each new cluster are irrevocably fixed when the cluster opens. Thus, the online algorithm maintains a collection of clusters, which is initially empty. Upon arrival of a new demand u_j, if u_j is not covered by an open cluster, the algorithm opens a new cluster $C(p, r)$ that includes u_j, and assigns u_j to it. The algorithm incurs an irrevocable cost of $f + r$ for the new cluster $C(p, r)$.

Competitive Ratio. We evaluate the performance of online algorithms using *competitive analysis*. A (randomized) algorithm is c-competitive if for any sequence of demand points, its (expected) cost is at most c times the cost of the optimal solution for the corresponding offline Sum-Radii instance. The (expected) cost of the algorithm is compared against the cost of an optimal offline algorithm that is aware of the entire demand sequence in advance and has no computational restrictions whatsoever.

Simplified Optimal. The following proposition simplifies the structure of the optimal solution in the competitive analysis.

Proposition 1. *Let S be a feasible solution of an instance $\mathcal{I}$ of OnlSumRad. Then, there is a feasible solution S' of $\mathcal{I}$ with a cost of at most twice the cost of S, where each cluster has a radius of $2^k f$, for some integer $k \geq 0$.*

Other Versions of OnlSumRad. For completeness, we discuss two seemingly less restricted versions of OnlSumRad, corresponding to the intermediate and the flexible model in [7]. In both versions, the demands are irrevocably assigned to a cluster upon arrival. In the Fixed-Radius version, only the radius of a new cluster is fixed when the cluster opens. The algorithms incurs an irrevocable cost of $f + r$ for each new cluster C of radius r. Then, new demands can be assigned to C, provided that $\mathrm{rad}(C) \leq r$. In the Flexible-Cluster version, a cluster C is a set of demands with neither a fixed center nor a fixed radius. The algorithm's cost for each cluster C is $f + \mathrm{rad}(C)$, where $\mathrm{rad}(C)$ may increase as new demands are added to C. The Fixed-Cluster version is a restriction of the Fixed-Radius version, which, in turn, is a restriction of the Flexible-Cluster version. The following proposition shows that from the viewpoint of competitive analysis, the three versions are essentially equivalent.

Proposition 2. *If there exists a c-competitive algorithm for the Fixed-Radius (resp. the Flexible-Cluster) version, then there exists a $2c$-competitive (resp. $10c$-competitive) algorithm for the Fixed-Cluster version.*

Parking Permit. In Parking Permit (ParkPermit), we are given a schedule of days, some of which are marked as driving days, and K types of permits, where a permit of each type k, $k = 1, \ldots, K$, has cost c_k and duration d_k. The goal is to purchase a set of permits of minimum total cost that cover all driving days. In the online setting, the driving days are presented one-by-one, and the algorithm irrevocably decides on the permits to purchase based on the driving days revealed so far.

Meyerson [17] observed that by losing a constant factor in the competitive ratio, we can focus on the *interval version* of ParkPermit, where each permit is available over specific time intervals (e.g. a weakly permit is valid from Monday to Sunday). Moreover, every day is covered by a single permit of each type k, and each permit

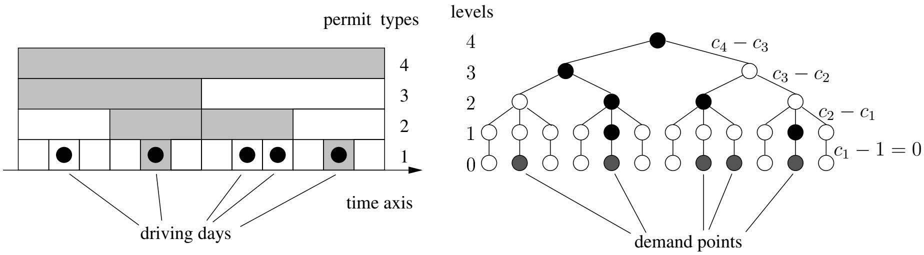


Fig. 1. An example of the reduction of Theorem 1. On the left, there is an instance of the interval version of ParkPermit. A feasible solution consists of the permits in grey. On the right, we depict the instance of OnlSumRad constructed in the proof of Theorem 1.

of type $k \geq 2$ has d_k/d_{k-1} permits of type $k-1$ embedded in it (see also Fig. 1). An interesting feature of the deterministic algorithm in [17] is that it is *time-sequence-independent*, in the sense that it applies, with the same competitive ratio of $O(K)$, even if the order in which the driving days are revealed may not be their time order (e.g. the adversary may mark Aug. 6 as a driving day, before marking May 25 as a driving day).

3 Online Sum-Radii Clustering and Parking Permit

In this section, we show that OnlSumRad in tree metrics and the interval version of ParkPermit are essentially equivalent problems. Our results either are directly based on this correspondence or exploit this correspondence so that they draw ideas from ParkPermit. We start with the following theorem, which shows that OnlSumRad in tree metrics is a generalization of the interval version of ParkPermit.

Theorem 1. *A c-competitive algorithm for Online Sum-Radii Clustering in HSTs with $K+1$ levels implies a c-competitive algorithm for the interval version of Parking Permit with K permit types.*

Proof. Given an instance $\mathcal{I}$ of the interval version of ParkPermit with K permit types, we construct an instance $\mathcal{I}'$ of OnlSumRad in an HST with $K+1$ levels such that any feasible solution of $\mathcal{I}$ is mapped, in an online fashion, to a feasible solution of $\mathcal{I}'$ of equal cost, and vice versa. Let $\mathcal{I}$ be an instance of the interval version of ParkPermit with K types of costs $c_1, \dots, c_K$ and durations $d_1, \dots, d_K$. Wlog., we assume that $c_1 = 1$ and that all days are covered by the permit of type K. Given the costs and the durations of the permits, we construct a tree T with appropriate edge lengths, which gives the metric space for $\mathcal{I}'$. The construction exploits the tree-like structure of the interval version (see also Fig. 1). Specifically, the tree T has $K+1$ levels, where the leaves correspond to the days of $\mathcal{I}$'s schedule, and each node at level k, $1 \leq k \leq K$, corresponds to a permit of type k.

Formally, the tree T has a leaf, at level 0, for each day in the schedule of $\mathcal{I}$. For each interval D_1 of d_1 days covered by a permit of type 1, there is a level-1 node v_1 in T whose children are the d_1 leaves corresponding to the days in D_1. The distance of each level-1 node to its children is $c_1 - 1 = 0$. Hence, opening a cluster $C(v_1, c_1 - 1)$ covers all leaves corresponding to the days in D_1. Similarly, for each interval D_k of d_k days covered by a permit of type k, $2 \leq k \leq K$, there is a node v_k at level k in T whose children are the d_k/d_{k-1} nodes at level $k-1$ corresponding to the permits of type $k-1$ embedded within the particular permit of type k. The distance of each level-k node to its children is $c_k - c_{k-1}$. Therefore, opening a cluster $C(v_k, c_k - 1)$ covers all leaves corresponding to the days in D_k. The cluster opening cost is $f = 1$. For each driving day t in $\mathcal{I}$, there is, in $\mathcal{I}'$, a demand located at the leaf of T corresponding to t.

Based on the correspondence between a type-k permit and a cluster $C(v_k, c_k - 1)$ rooted at a level-k node v_k, we can show that any solution of $\mathcal{I}$ is mapped, in an online fashion, to a solution of $\mathcal{I}'$ of equal cost, and vice versa. □

In the proof of Theorem 1, if the ParkPermit instance has $d_1 = 1$ and $c_k = 2^k$, for each type k, the tree T is essentially a 2-HST with K levels where all nodes at the same level k have d_k/d_{k-1} children. Thus, combined with Theorem 1, the following lemma shows that OnlSumRad in such tree metrics is similar to the interval version of ParkPermit. The proof of Lemma 1 applies the reverse reduction of Theorem 1.

Lemma 1. *A c-competitive time-sequence-independent algorithm for the interval version of ParkPermit with K permits implies a c-competitive algorithm for OnlSumRad in HSTs with K levels, where all nodes at the same level have the same number of children and all demands are located at the leaves.*

4 Lower Bounds on the Competitive Ratio

By Theorem 1, OnlSumRad in trees with $K+1$ levels is a generalization of ParkPermit with K permit types. Therefore, the results of [17] imply a lower bound of $\Omega(K)$ (resp. $\Omega(\log K)$) on the deterministic (resp. randomized) competitive ratio of OnlSumRad in trees with K levels. However, a lower bound on the competitive ratio of OnlSumRad would rather be expressed in terms of the number of demands n, because there is no simple and natural way of defining the number of "levels" of a general metric space, and because for online clustering problems, the competitive ratio, if not constant, is typically stated as a function of n.

Going through the proofs of Theorem 1 and of [17, Theorems 3.2 and 4.6], we can translate the lower bounds on the competitive ratio of ParkPermit, expressed as a function of K, into equivalent lower lower bounds for OnlSumRad, expressed as a function of n. In fact, the proofs of [17, Theorems 3.2 and 4.6] require that the ratio d_k/d_{k-1} of the number of days covered by permits of type k and $k-1$ is $2K$. Thus, in the proof of Theorem 1, the tree T has $(2K)^K$ leaves, and the number of demands n is at most $(2K)^K$. Combining this with the lower bound of $\Omega(\log K)$ on the randomized competitive ratio of ParkPermit [17, Theorem 4.6], we obtain the following corollary:

Corollary 1. *The competitive ratio of any randomized algorithm for Online Sum-Radii Clustering in tree metrics is $\Omega(\log\log n)$, where n is the number of demands.*

A Stronger Lower Bound on the Deterministic Competitive Ratio. This approach gives a lower bound of $\Omega(\frac{\log n}{\log\log n})$ on the deterministic competitive ratio of Online Sum-Radii Clustering. Using a ternary HST, we next obtain a stronger lower bound.

Theorem 2. *The competitive ratio of any deterministic online algorithm for Online Sum-Radii Clustering in tree metrics is $\Omega(\log n)$, where n is the number of demands.*

Proof. For simplicity, let us assume that n is an integral power of 3. For some constant $\alpha \in [2, 3)$, we consider an α-HST T of height $K = \log_3 n$ whose non-leaf nodes have 3 children each. The cluster opening cost is $f = 1$. Let A be any deterministic algorithm. We consider a sequence of demands located at the leaves of T. More precisely, starting from the leftmost leaf and advancing towards the rightmost leaf, the next demand in the sequence is located at the next leaf not covered by an open cluster of A. Since T has n leaves, A may cover all leaves of T before the arrival of n demands. Then, the demand sequence is completed in an arbitrary way that does not increase the optimal cost. We let C_{OPT} be the optimal cost, and let C_A be the cost of A on this demand sequence.

We let $c_k = 1 + \sum_{\ell=0}^{k-1} \alpha^\ell$ denote the cost of a cluster centered at a level-k node v_k with radius equal to the distance of v_k to the nearest leaf. We observe that for any $k \geq 1$ and any $\alpha \geq 2$, $c_k \leq \alpha c_{k-1}$. We classify the clusters opened by A according to their cost. Specifically, we let L_k, $0 \leq k \leq K$, be the set of A's clusters with cost in $[c_k, c_{k+1})$, and let $\ell_k = |L_k|$ be the number of such clusters. The key property is that a cluster in L_k can cover the demands of a subtree rooted at level at most k, but not higher. Therefore, we can assume that all A's clusters in L_k are centered at a level-k node and have cost equal to c_k, and obtain a lower bound of $C_A \geq \sum_{k=0}^{K} \ell_k c_k$ on the algorithm's cost.

To derive an upper bound on the optimal cost in terms of C_A, we distinguish between good and bad active subtrees, depending on the size of the largest radius cluster with which A covers the demand points in them. Formally, a subtree T_k rooted at level k is *active* if there is a demand point located at some leaf of it. For an active subtree T_k, we let $C_{T_k}^{\max}$ denote the largest radius cluster opened by A when a new demand point in T_k arrives. Let j, $0 \leq j \leq K$, be such that $C_{T_k}^{\max} \in L_j$. Namely, $C_{T_k}^{\max}$ is centered at a level-j node v_j and covers the entire subtree rooted at v_j. If $j \geq k$, i.e. if $C_{T_k}^{\max}$ covers T_k entirely, we say that T_k is a *good* (active) subtree (for the algorithm A). If $j < k$, i.e. if $C_{T_k}^{\max}$ does not cover T_k entirely, we say that T_k is a *bad* (active) subtree (for A).

For each $k = 0, \ldots, K$, we let g_k (resp. b_k) denote the number of good (resp. bad) active subtrees rooted at level k. To bound g_k from above, we observe that the last demand point of each good active subtree rooted at level k is covered by a new cluster of A rooted at a level $j \geq k$. Therefore, the number of good active subtrees rooted at level k is at most the number of clusters in $\bigcup_{j=k}^{K} L_j$. Formally, for each level $k \geq 0$, $g_k \leq \sum_{j=k}^{K} \ell_j$. To bound b_k from above, we first observe that each active leaf / demand point is a good active level-0 subtree, and thus $b_0 = 0$. For each level $k \geq 1$, we observe that if T_k is a bad subtree, then by the definition of the demand sequence, the 3 subtrees rooted at the children of T_k's root are all active. Moreover, each of these subtrees is either a bad subtree rooted at level $k - 1$, in which case it is counted in b_{k-1}, or a good subtree covered by a cluster in L_{k-1}, in which case it is counted in ℓ_{k-1}. Therefore, for each level $k \geq 1$, $3b_k \leq b_{k-1} + \ell_{k-1}$.

Using these bounds on g_k and b_k, we can bound from above the optimal cost in terms of C_A. To this end, the crucial observation is that we can obtain a feasible solution by opening a cluster of cost c_k centered at the root of every active subtree rooted at level k. Since the number of active subtrees rooted at level k is $b_k + g_k$, we obtain that for every $k \geq 0$, $C_{OPT} \leq c_k(b_k + g_k)$. Using the upper bound on g_k and summing up for $k = 0, \dots, K$, we have that $(K+1)C_{OPT} \leq \sum_{k=0}^{K} c_k b_k + \sum_{k=0}^{K} c_k \sum_{j=k}^{K} \ell_j$.

Using that $c_k \leq \alpha^k$ and that $c_k \leq \alpha c_{k-1}$, which hold for all $\alpha \geq 2$, we bound the second term by:

$$\sum_{k=0}^{K} c_k \sum_{j=k}^{K} \ell_j = \sum_{k=0}^{K} \ell_k \sum_{j=0}^{k} c_j \leq \sum_{k=0}^{K} \ell_k \sum_{j=0}^{k} \alpha^j \leq \sum_{k=0}^{K} \ell_k c_{k+1} \leq \alpha \sum_{k=0}^{K} \ell_k c_k \leq \alpha C_A$$

To bound the first term, we use that for every level $k \geq 1$, $3b_k \leq b_{k-1} + \ell_{k-1}$ and $c_k \leq \alpha c_{k-1}$. Therefore, $(3/\alpha) b_k c_k \leq (b_{k-1} + \ell_{k-1}) c_{k-1}$. Summing up for $k = 1, \dots, K$, we have that:

$$\frac{3}{\alpha} \sum_{k=1}^{K} b_k c_k \leq \sum_{k=1}^{K} b_{k-1} c_{k-1} + \sum_{k=1}^{K} \ell_{k-1} c_{k-1}$$

Using that $b_0 = 0$ and that $\alpha < 3$, we obtain that:

$$\frac{3}{\alpha} \sum_{k=0}^{K} b_k c_k \leq \sum_{k=0}^{K-1} b_k c_k + \sum_{k=0}^{K-1} \ell_k c_k \leq \sum_{k=0}^{K} b_k c_k + C_A \quad \Rightarrow \quad \sum_{k=0}^{K} b_k c_k \leq \frac{\alpha}{3-\alpha} C_A$$

Putting everything together, we conclude that for any $\alpha \in [2, 3)$, $(K+1)C_{OPT} \leq (\alpha + \frac{\alpha}{3-\alpha})C_A$. Since $K = \log_3 n$, this implies the theorem. □

A Lower Bound for Deterministic OnlSumRad on the Plane. Motivated by the fact that the deterministic competitive ratio of OnlSumRad on the line is constant [7], we study OnlSumRad in the Euclidean plane. The following theorem uses a constant-distortion planar embedding of a ternary α-HST, and establishes a lower bound of $\Omega(\log n)$ on the deterministic competitive ratio of OnlSumRad on the Euclidean plane.

Theorem 3. *The competitive ratio of any deterministic algorithm for Online Sum-Radii Clustering on the Euclidean plane is* $\Omega(\log n)$, *where* n *is the number of demands.*

Proof sketch. Using a planar embedding of a ternary α-HST T with distortion $D_\alpha \leq \sqrt{2}\,\alpha/(\alpha-2)$, we can show that a c-competitive algorithm for OnlSumRad on the plane implies a $2cD_\alpha$-competitive algorithm for HSTs. □

5 An Optimal Primal-Dual Online Algorithm

Next, we present a deterministic primal-dual algorithm for OnlSumRad in a general metric space (M, d). In the following, we assume that the optimal solution only consists of clusters with radius $2^k f$, where k is a non-negative integer (see also Proposition 1). For simplicity, we let $r_k = 2^k f$, if $k \geq 0$, and $r_k = 0$, if $k = -1$. Let $N = \mathbb{N} \cup \{-1\}$. Then, the following are a Linear Programming relaxation of OnlSumRad and its dual:

$$\begin{aligned} \min \quad & \sum_{(z,k)\in M\times N} x_{zk}(f + r_k) \\ \text{s.t} \quad & \sum_{(z,k):d(u_j,z)\leq r_k} x_{zk} \geq 1 \qquad \forall\, u_j \\ & x_{zk} \geq 0 \qquad \forall\,(z,k) \end{aligned}
\qquad
\begin{aligned} \max \quad & \sum_{j=1}^{n} a_j \\ \text{s.t} \quad & \sum_{j:d(u_j,z)\leq r_k} a_j \leq f + r_k \qquad \forall\,(z,k) \\ & a_j \geq 0 \qquad \forall\, u_j \end{aligned}$$

In the primal program, there is a variable x_{zk} for each point z and each $k \in N$ that indicates the extent to which cluster $C(z, r_k)$ is open. The constraints require that each demand u_j is fractionally covered. If we require that $x_{zk} \in \{0, 1\}$ for all z, k, we obtain an Integer Programming formulation of OnlSumRad. In the dual, there is a variable a_j for each demand u_j, and the constraints require that no potential cluster is "overpaid".

The primal-dual algorithm, or PD-SumRad in short, maintains a collection of clusters that cover all the demands processed so far. When a new demand u_j, $j = 1, \ldots, n$, arrives, if u_j is covered by an already open cluster C, PD-SumRad assigns u_j to C and sets u_j's dual variable a_j to 0. Otherwise, PD-SumRad sets a_j to f. This makes the dual constraint corresponding to $(u_j, -1)$ and possibly some other dual constraints tight. PD-SumRad finds the maximum $k \in N$ such that for some point $z \in M$, the dual constraint corresponding to (z, k) becomes tight due to a_j. Then, PD-SumRad opens a new cluster $C(z, 3r_k)$ and assigns u_j to it. The main result of this section is that:

Theorem 4. *The competitive ratio of PD-SumRad is* $\Theta(\log n)$.

The lower bound on the competitive ratio of PD-SumRad follows from Theorem 2. To establish the upper bound, we first observe that the dual solution maintained by PD-SumRad is feasible. Thus the optimal cost for any demand sequence is at least the value of the dual solution maintained by PD-SumRad. Combining this observation with the following lemma, which shows that the total cost of PD-SumRad is at most $O(\log n)$ times the value of its dual solution, we obtain the claimed competitive ratio.

Lemma 2. *The cost of PD-SumRad is at most* $3(2 + \log_2 n) \sum_{j=1}^{n} a_j$.

Proof. We call a cluster $C(z, r_k)$ *tight* if the dual constraint corresponding to (z, k) is satisfied with equality. We observe that for any integer $k > \log_2 n$ and for all points z, $C(z, k)$ cannot become tight, because the lefthand-side of any dual constraint is at most nf. Therefore, we can restrict our attention to at most $2 + \log_2 n$ values of k.

Next, we show that for every $k = -1, 0, \ldots, \lfloor \log_2 n \rfloor$, each demand u_j with $a_j > 0$ contributes to the opening cost of at most one cluster with radius $3r_k$. Namely, there is at most one cluster $C(z, 3r_k)$ for which u_j belongs to the tight cluster $C(z, r_k)$. For sake of contradiction, let us assume that for some value of k, PD-SumRad opens two clusters $C_1 = C(z_1, 3r_k)$ and $C_2 = C(z_2, 3r_k)$ for which there is some u_j with $a_j > 0$ that belongs to both $C(z_1, r_k)$ and $C(z_2, r_k)$. Since PD-SumRad opens at most one new cluster when a new demand is processed, one of the clusters C_1, C_2 opens before the other. So, let us assume that C_1 opens before C_2. This means that PD-SumRad opened C_1 in response to a demand $u_{j'}$, with $j' \leq j$, that was uncovered at its arrival time and made $C(z_1, r_k)$ tight. Then, any subsequent demand $u \in C(z_2, r_k)$ is covered by C_1, because:

$$d(u, z_1) \leq d(u, u_j) + d(u_j, z_1) \leq 2r_k + r_k = 3r_k$$

The second inequality above holds because both u and u_j belong to $C(z_2, r_k)$ and u_j also belongs to $C(z_1, r_k)$. Therefore, after C_1 opens, there are no uncovered demands in $C(z_2, r_k)$ that can force PD-SumRad to open C_2, a contradiction.

To conclude the proof of the lemma, we observe that when PD-SumRad opens a new cluster $C(z, 3r_k)$, the cluster $C(z, r_k)$ is tight. Hence, the total cost of $C(z, 3r_k)$ is at most $3\sum_{u_j \in C(z,r_k)} a_j$. Therefore, the total cost of PD-SumRad is at most:

$$\sum_{(z,k):C(z,3r_k)\,\text{op.}} \sum_{u_j \in C(z,r_k)} 3a_j = 3\sum_{j=1}^{n} a_j \,|\{(z,k) : C(z,3r_k)\,\text{opens} \wedge u_j \in C(z,r_k)\}|$$
$$\leq 3(2+\log_2 n)\sum_{j=1}^{n} a_j$$

The inequality holds because for each $k = -1, 0, \ldots, \lfloor \log_2 n \rfloor$ and each u_j with $a_j > 0$, there is at most one pair (z, k) such that $C(z, 3r_k)$ opens and $u_j \in C(z, r_k)$. □

6 Randomized and Fractional Online Algorithms

Throughout this section, we assume that the optimal solution only consists of clusters of radius $2^k f$. For simplicity, we assume that n is an integral power of 2 and known in advance. Using standard techniques, we can remove these assumptions, by only losing a constant factor in the competitive ratio.

Randomized Algorithm. We first present Simple-SumRad, that is a simple randomized algorithm of logarithmic competitiveness. Simple-SumRad is *memoryless*, in the sense that it keeps in memory only its solution, namely the centers and the radii of its clusters. When a new demand u_j arrives, if u_j is covered by an already open cluster C, Simple-SumRad assigns u_j to C. Otherwise, for each $k = 0, \ldots, \log_2 n$, the algorithm opens a new cluster $C(u_j, 2^k f)$ with probability 2^{-k}, and assigns u_j to the cluster $C(u_j, f)$, which opens with probability 1. We can show that:

Lemma 3. *Simple-SumRad achieves a competitive ratio of at most* $2(4 + \log_2 n)$.

Fractional Algorithm. We conclude with a deterministic $O(\log \log n)$-competitive algorithm for the fractional version of OnlSumRad in general metric spaces. The fractional algorithm is based on the primal-dual approach of [2,1], and is a generalization of the online algorithm for the fractional version of ParkPermit in [17, Section 4.1].

A fractional algorithm maintains, in an online fashion, a feasible solution to the Linear Programming relaxation of OnlSumRad. In the notation of Section 5, for each point-type pair (z, k), the algorithm maintains a fraction x_{zk}, which denotes the extent to which the cluster $C(z, r_k)$ opens, and can only increase as new demands arrive. For each demand u_j, the fractions of the clusters covering u_j must sum up to at least 1, i.e. $\sum_{(z,k):u_j \in C(z,r_k)} x_{zk} \geq 1$. The total cost of the algorithm is $\sum_{(z,k)} x_{zk}(f + r_k)$.

The Algorithm. The fractional algorithm, or Frac-SumRad in short, considers $K + 1$ different types of clusters, where $K = \log_2 n$. For each $k = 1, \ldots, K + 1$, we let $c_k = f + r_k$ denote the cost of a cluster $C(p, r_k)$ of type k. The algorithm considers only the demand locations as potential cluster centers. For convenience, for each demand u_j and for each k, we let x_{jk} be the extent to which the cluster $C(u_j, r_k)$ is open, with the understanding that $x_{jk} = 0$ before u_j arrives. Similarly, we let

$F_{jk} = \sum_{(i,k):u_j \in C(u_i,r_k)} x_{ik}$ be the extent to which demand u_j is covered by clusters of type k, and let $F_j = \sum_k F_{jk}$ be the extent to which u_j is covered.

When a new demand u_j, $j = 1, \ldots, n$, arrives, if $F_j \geq 1$, u_j is already covered. Otherwise, while $F_j < 1$, Frac-SumRad performs the following operation:

1. For every $k = 1, \ldots K + 1$, $x_{jk} \leftarrow x_{jk} + \frac{1}{c_k(K+1)}$
2. For every $k = 1, \ldots, K + 1$ and every demand $u_i \in C(u_j, r_k)$, $x_{ik} \leftarrow x_{ik}(1 + \frac{1}{c_k})$

Frac-SumRad maintains a (fractional) feasible solution in an online fashion. The proof of the following theorem extends the competitive analysis in [17, Section 4.1].

Theorem 5. *The competitive ratio of Frac-SumRad is* $O(\log \log n)$.

References

1. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: A General Approach to Online Network Optimization Problems. ACM Transactions on Algorithms 2(4), 640–660 (2006)
2. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: The Online Set Cover Problem. SIAM J. on Computing 39(2), 361–370 (2009)
3. Bilò, V., Caragiannis, I., Kaklamanis, C., Kanellopoulos, P.: Geometric Clustering to Minimize the Sum of Cluster Sizes. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 460–471. Springer, Heidelberg (2005)
4. Chan, T.M., Zarrabi-Zadeh, H.: A Randomized Algorithm for Online Unit Clustering. Theory of Computing Systems 45(3), 486–496 (2009)
5. Charikar, M., Chekuri, C., Feder, T., Motwani, R.: Incremental Clustering and Dynamic Information Retrieval. SIAM J. on Computing 33(6), 1417–1440 (2004)
6. Charikar, M., Panigrahy, R.: Clustering to Minimize the Sum of Cluster Diameters. J. of Computer and System Sciences 68(2), 417–441 (2004)
7. Csirik, J., Epstein, L., Imreh, C., Levin, A.: Online Clustering with Variable Sized Clusters. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 282–293. Springer, Heidelberg (2010)
8. Divéki, G., Imreh, C.: An Online 2-Dimensional Clustering Problem with Variable Sized Clusters. Submitted for publication (2011)
9. Doddi, S., Marathe, M.V., Ravi, S.S., Taylor, D.S., Widmayer, P.: Approximation Algorithms for Clustering to Minimize the Sum of Diameters. Nordic J. Computing 7(3), 185–203 (2000)
10. Ehmsen, M.R., Larsen, K.S.: Better Bounds on Online Unit Clustering. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 371–382. Springer, Heidelberg (2010)
11. Epstein, L., van Stee, R.: On the Online Unit Clustering Problem. ACM Transactions on Algorithms 7(1), 7 (2010)
12. Fotakis, D.: On the Competitive Ratio for Online Facility Location. Algorithmica 50(1), 1–57 (2008)
13. Gibson, M., Kanade, G., Krohn, E., Pirwani, I.A., Varadarajan, K.: On Clustering to Minimize the Sum of Radii. In: SODA 2008, pp. 819–815 (2008)
14. Gibson, M., Kanade, G., Krohn, E., Pirwani, I.A., Varadarajan, K.: On Metric Clustering to Minimize the Sum of Radii. Algorithmica 57, 484–498 (2010)
15. Lev-Tov, N., Peleg, D.: Polynomial Time Approximation Schemes for Base Station Coverage with Minimum Total Radii. Computer Networks 47(4), 489–501 (2005)
16. Meyerson, A.: Online Facility Location. In: FOCS 2001, pp. 426–431 (2001)
17. Meyerson, A.: The Parking Permit Problem. In: FOCS 2005, pp. 274–284 (2005)
18. Schaeffer, S.E.: Graph Clustering. Computer Science Review 1, 27–64 (2007)

Observe and Remain Silent (Communication-Less Agent Location Discovery)*

Tom Friedetzky[1], Leszek Gąsieniec[2], Thomas Gorry[2], and Russell Martin[2]

[1] School of Engineering and Computing Sciences, Durham University, UK
[2] Department of Computer Science, University of Liverpool, UK

Abstract. We study a randomised distributed communication-less coordination mechanism for n uniform anonymous agents located on a circle with unit circumference. We assume the agents are located at arbitrary but distinct positions, unknown to other agents. The agents perform actions in synchronised rounds. At the start of each round an agent chooses the direction of its movement (*clockwise* or *anticlockwise*), and moves at unit speed during this round. Agents are not allowed to overpass, i.e., when an agent collides with another it instantly starts moving with the same speed in the opposite direction. Agents cannot leave marks on the ring, have zero vision and cannot exchange messages. However, on the conclusion of each round each agent has access to (some, not necessarily all) information regarding its trajectory during this round. This information can be processed and stored by the agent for further analysis.

The *location discovery task* to be performed by each agent is to determine the initial position of every other agent and eventually to stop at its initial position, or proceed to another task, in a fully synchronised manner. Our primary motivation is to study distributed systems where agents collect the minimum amount of information that is necessary to accomplish this location discovery task.

Our main result is a fully distributed randomised (Las Vegas type) algorithm, solving the *location discovery problem* *w.h.p.* in $O(n \log^2 n)$ rounds (assuming the agents collect sufficient information). Note that our result also holds if initially the agents do not know the value of n and they have no coherent sense of direction.

1 Introduction

A cycle-based topology of communication networks is very often identified with the *ring* of discrete nodes in which each node has two neighbours at its opposite sides. The ring network is one of the most studied network topologies in the context of standard distributed computation tasks [2,19,22] as well as coordination mechanisms for mobile agents [18]. In this paper, however, we focus on geometric rings, later referred to as *circles*. The work presented in this paper refers to the recently popularised concept of *swarms*, i.e., large groups of fairly primitive but cost-effective entities (robots, agents) that can be deployed to perform an exploration or a monitoring task in a hard-to-access hostile environment. There has been substantial progress in the design of efficient distributed coordination mechanisms in a variety of models for mobile agents, e.g., see [1,5,16,23]. In this paper we consider a version of the model introduced in [1].

* This work was partially supported by the Royal Society Grant IJP - 2010/R2.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 407–418, 2012.

In that model, the agents operate in synchronised rounds, they are assumed to be anonymous, and they lack means of communication. An agent wakes up at the beginning of each round and performs its move that depends on the current location of other agents in the network. In the model from [1] the moves are assumed to be instantaneous, but this last assumption is not true for us here. Numerous algorithms have been developed in the literature for a variety of control problems for robot swarms, see [1,5,12,13,16,23]. Most of these algorithmic solutions, with certain exceptions, e.g., [6], impose on the participating agents access to the *global* picture, in other words the ability to monitor performance of all agents. While there is a large volume of agent network exploration algorithms, they mainly focus on network topology discovery either in graph-based networks [3,4,7,14] or in geometric setting [9,10,15,17].

In this paper, we focus on the network model similar to [1] in which communication is limited to a bare minimum. In such networks, the communication deficiency of an agent is compensated by an astute observation and analysis of its own movement. The *trajectory* of an agent's movement in a given round is represented as a continuous, rectifiable curve, that connects the start and the end points of the route adopted by the agent. While moving along their trajectories, agents collide with their immediate neighbours, and information on the exact location of those collisions might be recorded and further processed. When agents are located on a circle, thanks to its closed topology, each agent may eventually conclude on the relative location of all agents' initial positions, even given only limited information about its trajectory. This procedure, in turn, enables other distributed mechanisms based on full synchronisation including equidistant distribution along the circumference and optimal boundary patrolling scheme. Most of the models adopted in the literature on swarms assume that the agents are either almost or entirely oblivious, i.e., throughout the computation process the agents follow a very simple, rarely amendable, routine of actions. Oblivious algorithms have many advantages including striking simplicity and self-stabilisation [11] properties.

The Model. In this paper we adopt a geometric network model, i.e., a circle with circumference one, along which a number of agents move and interact in fully synchronised rounds (each of which lasts one unit of time). The agents are uniform and anonymous. Moreover, the agents do not necessarily share the same sense of direction, i.e., while each agent distinguishes between its own clockwise (C) and anticlockwise (A) directions, agents may not have a coherent view on this (see Section 3.4 for more on this). At the beginning of each round an agent chooses a direction of its move from $\{A, C\}$ and moves at unit speed. We assume that agents are not allowed to overpass each other along the circle. In particular, when an agent collides with another (agent) it instantly starts moving with the same speed but in the opposite direction. The agents cannot leave marks on the ring, they have zero visibility and cannot exchange messages. Instead, on the conclusion of each round every agent learns a specific information concerning its recent trajectory. In particular, for odd n we assume that an agent is informed about the relative distance between its location at the start and the end of this round. For even n, however, the agents must also learn about the exact time (location) of their first collisions during this round. This information can be processed or stored for further analysis. The aim of an agent is to discover the initial positions of all other agents. Our main motivation in this paper is to study distributed systems where agents collect the *minimum amount* of information necessary to accomplish the location discovery task, and the novelty in this paper comes from considering this situation where agents operate with a very limited amount of information collected during the discovery procedure. One might consider, for example, that an agent spends energy to determine its current location, and wants to minimize its energy expenditure.

Since the agents never overpass we may assume that the agents are arranged in an implicit (i.e., never disclosed to the agents) periodic order from a_0 to a_{n-1}. We also denote by p_i the original positions of a_i, for all $i \in [n]$[1]. Note that, due to the periodic order of agents, all calculations on implicit labels of agents that follow are performed modulo n.

Our Results. We assume that n mobile agents are initially located on the boundary of a circle at arbitrary, distinct and undisclosed positions. As stated previously, the task of each agent is to determine the initial position of every other agent. On the conclusion of the algorithm agents either synchronously stop at their initial positions or may proceed with another task. For the clarity of presentation, we first provide a solution to the distributed location discovery under assumption that the agents have a coherent sense of direction and the value n is known in advance to all agents. Later, in Sections 3.3 and 3.4 we provide further evidence on how these two assumptions can be dropped. Finally, we briefly describe how the location-discovery mechanism can be used to coordinate actions of agents in distributed boundary patrolling on circles, see Section 3.5.

This last part should be seen as a natural continuation of [8] devoted to efficient centralised patrolling mechanisms designed for agents with distinct maximum speeds. We believe, this is the first attempt to solve the distributed boundary patrolling problem in the geometric ring (circle) model.

All our bounds hold with high probability[2] (*w.h.p.*) for n large enough. However, one can easily modify our solutions such that by periodically repeating actions of agents, they can solve the task with the required level of confidence even for smaller, e.g., constant values of n.

2 Rotation Mechanism

The location-discovery algorithm is formed of a number of *stages*. Each stage is a sequence of at most n consecutive rounds, each of unit duration. At the beginning of the first round of each stage an agent a_i randomly chooses the direction (clockwise or anticlockwise) of its movement, and moves with unit speed throughout the entire stage. Later throughout the same stage, the exact location and the movement direction of a_i depends solely on the collisions with its neighbours a_{i-1} and a_{i+1}. We show that on conclusion of each round the agents always reside at the initial positions $p_0, \ldots, p_{n-1}$, where there is a $k \in [n]$ such that the current location of agent a_i corresponds to p_{i+k}. Note that this observation allows agents to visit (and record) the initial positions of other agents. Thus, part of the limited amount of information that an agent obtains is its

[1] $[n] = \{0, 1, \ldots, n-1\}$ for any natural number n.

[2] With probability at least $1 - 1/n^c$ for some positive constant c.

position, *relative to its initial starting location*, at the end of each round. A stage concludes at the end of a round when each agent a_i arrives at its original starting position p_i. We show that *w.h.p.* agents require $O(\log^2 n)$ stages to learn the locations of their counterparts. Since each stage is formed of at most n rounds, the total complexity of our algorithms is bounded by $O(n \log^2 n)$.

Throughout the discovery procedure, agents move with uniform speed one. Recall when two agents collide, they instantly bounce back without changing their uniform unit speed. While observing two indistinguishable colliding agents, one could wrongly conclude that the two agents overpass each other. We assume that at the beginning of each stage of our algorithm every agent a_i holds a unique *virtual baton* b_i. During the first collision with either a_{i-1} or a_{i+1} this baton gets exchanged for a baton currently held by the respective agent. In due course, further exchanges of batons take place. We emphasize that the concept of batons is solely a proof device in what follows, that they do not actually exist as far as the agents are concerned, and that no actual communication (or exchange of any object) takes place between the agents when they collide.

Lemma 1. *At the start of each round baton b_i resides at position p_i, for all $i \in [n]$.*

Proof. At the start of the location discovery procedure, b_i resides at p_i for all i. During the first round, baton b_i moves in a unidirectional manner with unit speed (being exchanged as appropriate during collisions), so b_i must arrive at p_i on the conclusion of this first round. Inductively, at the end of each round (i.e. start of the next round) of the procedure, b_i will reside at position p_i. □

Using Lemma 1 we can conclude that at the start of each round the agents populate initial locations $p_0, \ldots, p_{n-1}$. In fact, one can state a more accurate lemma.

Lemma 2. *There is a $k \in [n]$ s.t. at the start of each round, for all $i \in [n]$, agent a_i resides at position p_{i+k}.*

Proof. At the start of the location discovery procedure, all initial positions are populated by the agents, each carrying a (virtual) baton. From Lemma 1, b_i begins (and ends) each round at position p_i. Since some agent must always be carrying b_i, there is an agent occupying the location p_i at the beginning of a round, and some (possibly different) agent occupying p_i (and holding b_i) at the end of the round. The same argument holds for each i, hence all n initial locations are occupied at the end (start) of each round. Recall that the agents never overpass, i.e., agent a_i always has the same neighbours a_{i-1} and a_{i+1}. Thus, if a_i resides at position p_{i+k} for some k, then a_{i-1} and a_{i+1} must reside at the respective locations p_{i+k-1} and p_{i+k+1}. □

Using the observation from Lemma 2, consider the respective locations p_{j+k_1} and p_{j+k_2} of agent a_i at the start of two consecutive rounds. One can conclude that during one round all agents rotated along the initial positions by a *rotation index* of $r = k_2 - k_1$, i.e. each agent experiences the same shift by r places (either clockwise or anticlockwise) between the beginning and the end of one round.

Lemma 3. *During one stage the rotation index r remains unchanged.*

Proof. At the beginning of a stage, the (random) choices of the agents determine the directions of the batons during the entire stage, i.e. if agent a_i chooses "clockwise", then baton b_i will move clockwise during that entire stage. Since at the beginning of each round, the virtual batons reside in their original positions (Lemma 1), and they don't change their directions during the entire stage, this means the pattern of movement and collisions (swaps of batons) of the agent beginning a round at p_i will be identical that of a_i during the first round of the stage. Hence, the rotation index remains unchanged during an entire stage. □

We now show the rotation index r depends on the initial choice of random directions adopted by the agents. Consider the first round of any stage. Let sets B_C and B_A contain the virtual batons that move during this round in the clockwise and anticlockwise directions, respectively, where $|B_C| = n_c$, $|B_A| = n_a$, and $n_c + n_a = n$. We say that during this stage virtual batons form a (n_c, n_a)-*configuration*.

Lemma 4. *In a stage with an (n_c, n_a)-configuration, the rotation index $r = n_c - n_a$.*

Proof. By Lemma 2 it is enough to prove the thesis of the lemma for one agent. Without loss of generality, assume that baton b_i is in B_C. At the beginning of any round baton b_i is aligned with position p_i, and assume that at the beginning of the considered round b_i is carried by agent a_j.

First note that b_i can be only exchanged with batons from B_A since all batons in B_C move with the same speed in the clockwise direction. Moreover, during any round every baton from B_C is exchanged with every baton from B_A exactly twice at certain antipodal points of the ring. Why is this? Suppose $b_k \in B_A$, and let d denote the distance (along the circumference) between b_i and b_k, measured in the clockwise direction. Note that $d < 1$ since agents start at distinct locations. Then b_i and b_j meet (are exchanged by colliding agents) at time $d/2$. After additional time $1/2$ (since $d/2 + 1/2 < 1$), b_i and b_k meet again at the antipodal point of their first collision before returning to their respective positions at p_i and p_k.

Thus, during any round baton b_i is exchanged between colliding agents exactly $2n_a$ times. Also, since b_i moves in the clockwise direction during each exchange, an index of the new hosting agent is increased by one. Thus at the end of the considered round when b_i arrives at p_i it is hosted by agent a_{j+2n_a}. This leads to conclusion that the rotation index is $r = -2n_a$.

Focusing on batons from B_A, one can use an analogous argument to prove the rotation factor $r = 2n_c$. Now since $n_c + n_a = n$ we get $-n_a = n_c (\mathrm{mod}\, n)$ and finally $-2n_a = 2n_c (\mathrm{mod}\, n)$ admitting the uniform rotation index r across all agents.

Finally, we add $n_a + n_c$ (that has value 0 modulo n) to $-2n_a$ and we obtain the rotation index $r = n_c - n_a$. □

3 The Location Discovery Algorithm

Using the thesis from Lemma 4, one can observe that if the rotation factor $n_c - n_a$ is relatively prime with n, denoted $\gcd(n_c - n_a, n) = 1$, a single stage with an (n_c, n_a)-configuration will last exactly n rounds. Moreover, during such a stage every agent will visit the original positions of all other agents. For example, if $n > 2$ is a prime number, one stage with $n_c, n_a \neq 0$ would be enough to discover the original positions of all agents. However, the situation complicates when n is a composite number. For example, when n is even, the difference $n_c - n_a$ is always even, meaning that n and $n_c - n_a$ cannot be relatively prime. This means that the mechanism described above will allow agents to discover at most half of the original positions.

In what follows we present first the discovery algorithm for odd values of n. Later we show how this algorithm can be amended to perform discovery also for even values of n.

3.1 Algorithm for Odd Values of n

As we already know, the algorithm works in stages concluded by agents' arrival to their initial positions. We again assume that the agents know n and they have a coherent sense of direction.

The algorithm explores the basic properties of the network model, reflected in the functionality of the procedure SINGLE-ROUND, accompanied by a randomised control mechanism. The procedure SINGLE-ROUND describes the performance of an agent during a single round. As input the procedure accepts two parameters: current relative location *loc* and $dir \in \{C, A\}$, i.e., the clockwise (C) or the anti-clockwise (A) direction of movement. On the conclusion of the round the procedure returns two parameters: *new-dir*, i.e., the direction of the agent to move in the new round; and a real value *new-loc*, a relative distance (positive or negative) that describes the position relative to its starting point at the beginning of the round. (This allows the agent to compute its relative distance from its starting point at the beginning of the stage, or the entire discovery procedure.) Recalling the discussion at the beginning of this section, the set of *new-loc* data collected during the procedure is sufficient to accomplish the location discovery task when n is odd, if the agents are in an (n_c, n_a)-configuration with $\gcd(n_c - n_a, n) = 1$.

The main (randomised) control mechanism of the procedure DISCOVER is presented in Figure 1. Initially, the list of known points is empty. At the end of each round the content of the list is updated. Notice that in step (3) the initial directions are being chosen uniformly at random as this clearly is the only sensible choice.

We say that a stage is *successful* when $\gcd(n_c - n_a, n) = 1$, i.e., when every agent visits all initial positions of other agents.

Lemma 5. *For any odd* $n > 0$, *a successful stage occurs within the first* $O(\log^2 n)$ *stages,* w.h.p.

Proof. We already know we target a distribution of directions with $\gcd(n_c - n_a, n) = 1$. To simplify our task we focus only on prime values of the rotation index where $|n_c - n_a| < \sqrt{n}$. Note that the probability that during a stage the value $|n_c - n_a|$ is obtained is $2 \cdot \binom{n}{n_c} / 2^n$. Using Stirling's factorial approximation one can prove that this probability is $\Omega(1/\sqrt{n})$, for all $|n_c - n_a| < \sqrt{n}$.

We also know [20] that for an integer m large enough ($> 15,985$) the m-th prime number is not larger than $m(\log m + \log\log m)$. This can be also interpreted that for m

```
procedure DISCOVER($n$: integer): list of points;
    the-list $\leftarrow \emptyset$;
    repeat      /* stage begins */
        pick direction $dir$ from $\{C, A\}$ uniformly in random;
        set $loc = 0$;
        repeat
            ($new\text{-}dir$,$new\text{-}loc$) $\leftarrow$ SINGLE-ROUND($loc, dir$);
            $the\text{-}list \leftarrow the\text{-}list \cup \{new\text{-}loc\}$;
            $dir \leftarrow new\text{-}dir$; $loc \leftarrow loc + new\text{-}loc$;
        until (loc = 0);          /* return to initial location, stage ends */
    until ($|the\text{-}list| = n$);        /* all points discovered, algorithm ends */
    return ($the\text{-}list$);
end.
```

Fig. 1. The location-discovery procedure of an agent

large enough there are $\Omega(m/\log m)$ prime numbers smaller than m. In particular, we can conclude that there are $\Omega(\sqrt{n}/\log n)$ primes between 0 and $\sqrt{n}$. Note, however that not all of these prime numbers need be relatively prime to n. However, n can have at most $O(\log n)$ prime divisors. So there are $\Omega(\frac{\sqrt{n}}{\log n} - \log n)$ primes between 0 and $\sqrt{n}$ that are also relatively prime to n.

This leads to the conclusion that the probability that any one stage is successful is $\Omega((\frac{\sqrt{n}}{\log n} - \log n) \cdot \frac{1}{\sqrt{n}}) = \Omega(\frac{1}{\log n})$. In other words, there exists a constant $c_o > 0$ such that a stage is successful with probability at least $c_o/\log n$.

Finally, the performance of DISCOVERY can be described by a Bernoulli process where the probability of success is $c_o/\log n$ in each stage. It is a well-known fact that after $c_o \log n$ stages of such a process, the probability of reaching a successful stage is constant, and after $c_o \log^2 n$ stages this probability is high. □

Since each stage is composed of at most n rounds, we have this conclusion.

Theorem 1. *For any large enough odd n, the number of rounds required to perform full discovery of the agents' initial positions is* $O(n \log^2 n)$ w.h.p.

3.2 Amendment for Even n

Note that when n is even, for any $n_c + n_a = n$ we have that $n_c - n_a$ is also even. Thus, one cannot await a stage with $\gcd(n_c - n_a, n) = 1$. Instead, we will target stages with $\gcd(n_c - n_a, n) = 2$, and in particular when $|n_c - n_a|$ is a double of a prime. Using a similar argument as in Lemma 5, one can prove that such a successful stage (where $\gcd(n_c - n_a, n) = 2$) occurs with probability $\Omega(1/\log n)$. In a successful stage, the agents form a bipartition $X_{\text{even}} \cup X_{\text{odd}}$, where $X_{\text{even}} = \{a_0, a_2, \ldots, a_{n-2}\}$ and $X_{\text{odd}} = \{a_1, a_3, \ldots, a_{n-1}\}$, and each agent learns the initial positions of all other agents in the same partition.

So can we solve the full location discovery problem in this case? Well, we can, provided an agent receives the *new-loc* data at the end of each round, as well as the time until (or location of) its *first* collision in each round. We show that this very limited additional information suffices to allow the agents to solve the discovery problem. Note that this amendment is not changing the model as defined in the Introduction, but merely changing the *amount* (and type) of information an agent receives during execution of the procedure.

Consider a successful stage where $\gcd(n_c - n_a, n) = 2$. During this stage, for any $i = 0, \ldots, n-1$ and $j = 0, \ldots, n/2$, agent a_{i+2j} (respectively, a_{i+2j+1}) also visits the initial position p_i (resp. p_{i+1}) of a_i (resp. a_{i+1}). Note that if at the beginning of this stage agent a_i picks direction C and agent a_{i+1} (from the other partition) picks direction A, the two agents meet halfway between p_i and p_{i+1} after traversing distance $\mathit{min\text{-}dist} = |p_i - p_{i+1}|/2$. When this happens, the agents can retrieve the original positions of one another, i.e. agent a_i concludes that $p_{i+1} = p_i + 2 \cdot \mathit{min\text{-}dist}$ and agent a_{i+1} concludes that $p_i = p_{i+1} - 2 \cdot \mathit{min\text{-}dist}$.

Note also that when agents a_i and a_{i+1} pick the same direction C (or A) the distance to the first meeting with a_{i+1} that is observed by agent a_i is always longer than *min-dist*. Thus, to learn the correct distance to p_{i+1}, a single successful stage with initial directions C of a_i and A of a_{i+1} is sufficient. In other words, we need to run the procedure DISCOVERY long enough to ensure that such a stage occurs *w.h.p.* This means that an agent $a_i \in X_{\text{odd}}$ will maintain a record of its current *estimates* of the starting locations of neighbors in X_{even}, and similar for an agent $a_i \in X_{\text{odd}}$. These estimates use the first collision information that an agent receives in each round, and the calculations described above (i.e. the first collision distance is used to estimate *min-dist* to the left or right neighbour). This record is updated, as appropriate, throughout the discovery procedure to build up a complete picture of the starting locations of the agents in the other partition.

Observe that if in the first round of a stage a_i (respectively, a_{i+1}) learns p_{i+1} (resp. p_i), then in the next $\frac{n}{2} - 1$ rounds of this stage every other agent a_{i+2j} (resp. a_{i+2j+1}) learns p_{i+1} (resp. p_i) since the directions of the batons b_i and b_{i+1} remain unchanged throughout the entire stage.

This leads us to conclusion that to solve the location-discovery problem for even n we need to run procedure DISCOVERY until, for each $i = 0, \ldots, n-1$, there is some successful stage in which agents a_i and a_{i+1} start moving during the first round in directions C and A, respectively. We show that $O(\log^2 n)$ stages of procedure DISCOVERY (modified so that an agent also collects the distance until its first collision in each round) still guarantee a solution to the discovery problem (*w.h.p.*) for even n.

Lemma 6. *For any even $n > 0$, each agent learns the positions of the others within the first $O(\log^2 n)$ stages* w.h.p.

Proof. In order to simplify the proof we focus on two sets of pairs of initial positions: $P_0 = \{(p_{2j}, p_{2j+1}) : j \in [\frac{n}{2} - 1]\}$ and $P_1 = \{(p_{2j+1}, p_{2j+2})) : j \in [\frac{n}{2} - 1]\}$. Within each set, each pair contains distinct agents' initial positions, and every such position belongs to some pair.

We split consecutive successful stages (with $\gcd(n_c - n_a, n) = 2$) of procedure DISCOVERY into two alternating sequences S_0 and S_1, where in stages from S_0 we consider pairs from P_0 and in stages from S_1 we consider pairs from P_1.

Without loss of generality, consider the sequence S_0. Recall that in these stages every agent visits every second initial position on the circle. Thus if, e.g., in the beginning of the first round of this stage agent a_{2j} moves in direction C and agent a_{2j+1} moves in direction A, after the first stage these two agents learn their relative positions, and in the remaining $\frac{n}{2}-1$ rounds of this stage all other agents in X_{even} learn p_{2j+1} *and* all other agents in X_{odd} learn p_{2j}. Thus we need to consider enough number of stages in S_0 such that, for each $j=0,\ldots,\frac{n}{2}$, there is a stage in which agent a_{2j} starts moving in direction C and agent a_{2j+1} starts moving in direction A.

We first assume that $|n_c - n_a| < \sqrt{n}$, which occurs *w.h.p.* Under this assumption, during each stage in S_0, we randomly populate the $\frac{n}{2}$ pairs in P_0 with pairs of directions, $(A,A),(A,C),(C,A)$ and (C,C). Since our primary interest is in the pair (C,A), we first estimate from below the expected number of (C,A) generated during each successful stage in S_0. We generate pairs sequentially at random assuming that initially the number of As and Cs is at least $\frac{n}{2}-\frac{\sqrt{n}}{2}$. We generate these pairs until either the remaining number of As or the remaining number of Cs is smaller than $\frac{n}{4}-\frac{\sqrt{n}}{2}$. This means that we generate at least $\frac{n/4}{2}=\frac{n}{8}$ pairs. One can now show that the probability of picking a mixed pair (C,A) is at least $1/5$.

Recall the *Coupon Collector's Problem* (CCP) in which one player must collect m coupons. During each consecutive attempt the player draws each coupon with probability $\frac{1}{m}$. One can use a short calculation and a union bound to prove that after $\alpha \cdot m \log m$ attempts the player is left without a full set of coupons with probability at most $\frac{1}{m^{\alpha-1}}$, for any constant $\alpha > 1$ [21]. CCP can be also executed in consecutive stages, where each stage can be formed of a fixed number ℓ of consecutive attempts. In this case one can conclude that it is enough to run $\alpha \cdot \frac{m}{\ell} \log m$ stages to collect all coupons with high probability $1-1/m^{\alpha-1}$.

We note here that random generation and further distribution of $(C,A)s$ in successful stages can be also seen as a version of coupon collection executed in stages. The $\frac{n}{2}$ pairs of positions in P_0 correspond to coupons in our version of CCP. During a single attempt in a successful stage (that occurs with probability $c_e/\log n$) a pair (C,A) is drawn with probability $1/5$ and allocated at random to one of the $n/2$ pairs in P_0. Thus, in a successful stage, in a single attempt each coupon (pair in P_0 with allocated (C,A)) is drawn with probability $2/(5n)$.

Compare now a single stage in standard CCP with a successful stage in our version of CCP. If in CCP a specific coupon is drawn more than once, the second and further attempts are void. In other words, these multiple attempts are wasted. In a valid stage of version of CCP (based on DISCOVERY), however, if an attempt results in a coupon (pair in P_0 with allocated (C,A)) that has been already collected in this stage, the attempt is continued until a not yet collected coupon is found. In other words, during a valid stage we may in fact generate more (but certainly not fewer) coupons compared to the respective stage in standard CCP.

Recall that during a successful stage at least $n/8$ sequential attempts are made, where each coupon out of $n/2$ is drawn with the probability $2/(5n)$. Since the probability defined on all coupons does not sum up to one, we may add a missing number of "null" coupons, each also drawn with probability $2/(5n)$. In turn, we obtain our version of CCP run in stages with $m=5n/2$ coupons and stages of length $\ell=\frac{n}{8}$. Recall also that in our version of CCP $\alpha \cdot \frac{m}{\ell} \log m$ stages are required to collect all coupons *w.h.p.* $1 - \frac{1}{m^{\alpha-1}}$. Since the length of each stage is $\ell = \frac{n}{8}$ and $m = \frac{5n}{2}$, one can conclude that $\alpha \cdot \frac{5n/2}{n/8} \log(\frac{5n}{2}) = 20\alpha \cdot (\log(\frac{5}{2}) + \log n) < 25\alpha \log n$ stages of DISCOVERY are needed to generate (C,A) for each pair in P_0, with probability $1 - 1/(\frac{5}{2} \cdot n)^{\alpha-1}$. This gives a high probability of success for $\alpha > 2$.

Similarly, one can analyse the generation of (C,A)s for all pairs in P_1. Thus *w.h.p.*, all agents can learn the position of all other agents with $O(\log^2 n)$ stages of DISCOVERY. □

3.3 Amendment for Unknown n

When n is unknown we will need another important observation. Consider a single stage where the direction of each of n batons is chosen uniformly at random. One can show that *w.h.p.* the batons form a (n_c, n_a)-configuration satisfying $|n_c - n_a| < 10\sqrt{n}$.

During the DISCOVERY procedure, each agent is constructing a (partial) map of the initial positions of all agents. If $\gcd(n_c - n_a, n) = 1$, then in one stage each agent will learn the initial positions of all other agents (but, of course, will not know that if n is unknown). If we have $\gcd(n_c - n_a, n) > 1$, an agent visits (records the location of) at least $\frac{n}{|n_c - n_a|}$ initial agent positions during the stage. Assuming the agent collects the distance to first collision in each round, it also builds (or updates) estimates of positions of nearest neighbours in its map (when n is odd, these may coincide with positions the agent visits; when n is even these estimates are necessary to determine the entire map, as outlined in Section 3.2).

Because $|n_c - n_a| < 10\sqrt{n}$ *w.h.p.*, we note that (*w.h.p.*) an agent will visit at least $\sqrt{n}/10$ initial positions in any stage. Hence, after an initial small (constant) number of stages, an agent can use the number of positions visited to obtain a very good estimate (or overestimate) for n, except that it may not know if n is even or odd. An agent determines the parity of n during the DISCOVERY procedure by observing if it actually visits the (calculated) positions of its nearest neighbours.

3.4 Sense of Direction Agreement

While a coherent sense of direction was not essential during the execution of procedure DISCOVERY, we will need it to solve other problems, such as boundary patrolling where all agents are asked to move in one direction. Recall that when the agents start they may not share the same sense of direction.

Two types of stages (rounds) can be distinguished: (1) when the agents do not collide, and (2) when collisions happen. Note that an extra agreement procedure is not needed if a stage of type (1) occurs. When the agents do not collide (i.e. after one time unit they arrive back at their starting locations) they assume that their current direction is the clockwise direction.

Observe that the probability that all agents choose the same direction is very small ($\frac{2}{2^n} = \frac{1}{2^{n-1}}$). Therefore we focus on stages of type (2) where we also have the rotation index $r \neq 0$. When the *first* such stage occurs, the agents that experience $r > 0$ do not change their understanding of the clockwise direction, and those with $r < 0$ replace the clockwise direction with its anticlockwise counterpart.

The probability that a stage of type (2) with $r \neq 0$ occurs is $1 - \frac{1}{2^{n-1}}$ when n is odd and $1 - \binom{n}{n/2}/2^n - \frac{1}{2^{n-1}} \gg 1/2$ when n is even. Thus after $O(\log^2 n)$ stages of procedure DISCOVERY the probability that agreement on sense of direction is reached is very high.

3.5 Equidistant Distribution and Boundary Patrolling

In this section we consider application of location discovery to the boundary patrolling problem, see [8], where agents walk along the circle indefinitely with the goal of minimising the maximum time between two consecutive visits at any point located on the circle. During the location discovery procedure, agents always move with the unit speed. However, in order to obtain the equidistant distribution each agent must be able to set its speed as a value $0 \leq s \leq 1$ during the course of a single round.

Recall that on the conclusion of procedure DISCOVERY every agent a_i is aware of the relative location of (or distance to) other agents. For equidistant distribution, during a single *adjustment round* each agent a_i is asked to move from p_i to a target position t_i, where $|t_{i+1} - t_i| = \frac{1}{n}$, for all $i \in [n]$.

Due to space limitations, we present only the main result of this section.

Theorem 2. *In the considered distributed model of computation, n agents with the maximum speed one can reach equidistant distribution in one round, after the location of all agents have been discovered.*

In the *boundary patrolling problem* the mobile agents are asked to adopt movement trajectories such that the maximum time, taken across all points in space, between two consecutive visits by some (possibly different) agents is minimised. We omit the proof of the following theorem due to space limitations.

Theorem 3. *In the considered distributed model of computation, n agents with the maximum speed one can adopt an optimal cyclic boundary patrolling strategy.*

4 Conclusion

We presented a fully distributed randomised algorithm that solves the location discovery problem *w.h.p.* in $O(n \log^2 n)$ rounds. We have shown that this result is also true if initially the agents are not aware of their number n and they have no coherent sense of direction.

Note however, that the circumference of the circle has to be known in advance. Otherwise, a participating agent might not be able to tell the difference between $n = 1$ and $n > 1$. In particular, if the agent imposed a limit on the traversal time until the first collision, the adversary would always choose the circumference to be large enough to accommodate distant locations between the agents preventing them from ever getting close enough. On the other hand, if the agent continues its search indefinitely, the adversary could choose $n = 1$ and the location discovery process would never end. Thus, it is important to know either n or the circumference of the circle.

References

1. Ando, H., Suzuki, I., Yamashita, M.: Formation and agreement problems for synchronous mobile robots with limited visibility. In: Proc. IEEE Symposium on Intelligent Control, pp. 453–460 (1995)
2. Attiya, H., Welch, J.: Distributed Computing. McGraw-Hill (1998)
3. Bender, M.A., Slonim, D.: The power of team exploration: Two robots can learn unlabeled directed graphs. In: Proc. 35th Annual Symposium on Foundations of Computer Science, FOCS 1994, pp. 75–85 (1994)
4. Chalopin, J., Flocchini, P., Mans, B., Santoro, N.: Network Exploration by Silent and Oblivious Robots. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 208–219. Springer, Heidelberg (2010)
5. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Solving the Robots Gathering Problem. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1181–1196. Springer, Heidelberg (2003)
6. Cohen, R., Peleg, D.: Local spreading algorithms for autonomous robot systems. Theoretical Computer Science 399(1-2), 71–82 (2008)
7. Cooper, C., Frieze, A., Radzik, T.: Multiple Random Walks and Interacting Particle Systems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 399–410. Springer, Heidelberg (2009)
8. Czyzowicz, J., Gąsieniec, L., Kosowski, A., Kranakis, E.: Boundary Patrolling by Mobile Agents with Distinct Maximal Speeds. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 701–712. Springer, Heidelberg (2011)
9. Czyzowicz, J., Labourel, A., Pelc, A.: Optimality and competitiveness of exploring polygons by mobile robots. Information and Computation 209(1), 74–88 (2011)
10. Deng, X., Kameda, T., Papadimitriou, C.: How to learn an unknown environment I: the rectilinear case. Journal of ACM 45(2), 215–245 (1998)
11. Dolev, S.: Self-Stabilization. MIT Press (2000)
12. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In: Aggarwal, A.K., Pandu Rangan, C. (eds.) ISAAC 1999. LNCS, vol. 1741, pp. 93–102. Springer, Heidelberg (1999)
13. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Pattern formation by autonomous robots without chirality. In: SIROCCO 2001, pp. 147–162 (2001)
14. Fraigniaud, P., Gąsieniec, L., Kowalski, D.R., Pelc, A.: Collective tree exploration. Networks 48(3), 166–177 (2006)
15. Hoffmann, F., Icking, C., Klein, R., Kriegel, K.: The polygon exploration problem. SIAM J. Computing 31(2), 577–600 (2001)
16. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theoretical Computer Science 390, 27–39 (2008)
17. Kong, C.S., Peng, N.A., Rekleitis, I.: Distributed coverage with multi-robot systems. In: Proc. Robotics and Automation, pp. 2423–2429 (2006)
18. Kranakis, E., Krizanc, D., Markou, E.: The Mobile Agent Rendezvous Problem in the Ring. Morgan and Claypool Publishers (2010)
19. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers (1996)
20. Massias, J.-P., Robin, G.: Bornes effectives pour certaines fonctions concernant les nombres premiers. Journal de Théorie des Nombres de Bordeaux 8, 215–242 (1996)
21. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
22. Santoro, N.: Design and Analysis of Distributed Algorithms. Wiley (2006)
23. Sugihara, K., Suzuki, I.: Distributed algorithms for formation of geometric patterns with many mobile robots. J. Robotic Systems 13(3), 127–139 (1996)

When Trees Grow Low: Shrubs and Fast MSO_1

Robert Ganian[1], Petr Hliněný[2], Jaroslav Nešetřil[3], Jan Obdržálek[2],
Patrice Ossona de Mendez[4], and Reshma Ramadurai[2]

[1] Institute for Informatics, Goethe University, Frankfurt, Germany
ganian@fi.muni.cz
[2] Faculty of Informatics, Masaryk University, Brno, Czech Republic*
{hlineny,obdrzalek,ramadur}@fi.muni.cz
[3] Computer Science Inst. of Charles University (IUUK), Praha, Czech Republic*,**
nesetril@kam.ms.mff.cuni.cz
[4] CNRS UMR 8557, École des Hautes Études en Sciences Sociales, Paris, France**
pom@ehess.fr

Abstract. Recent characterization [9] of those graphs for which coloured MSO_2 model checking is fast raised the interest in the graph invariant called *tree-depth*. Looking for a similar characterization for (coloured) MSO_1, we introduce the notion of *shrub-depth* of a graph class. To prove that MSO_1 model checking is fast for classes of bounded shrub-depth, we show that shrub-depth exactly characterizes the graph classes having interpretation in coloured trees of bounded height. We also introduce a common extension of cographs and of graphs with bounded shrub-depth — *m-partite cographs* (still of bounded clique-width), which are well quasi-ordered by the relation "is an induced subgraph of" and therefore allow polynomial time testing of hereditary properties.

1 Introduction

In this paper, we are interested in graph parameters that are intermediate between clique-width and tree-depth, sharing the nice properties of both. Clique-width, defined in [4], is the older of the two notions. In several aspects, the theory of graphs of bounded clique-width is similar to the one of bounded tree-width. Indeed, bounded tree-width implies bounded clique-width. However, unlike tree-width, graphs with bounded clique-width include arbitrarily large cliques and other dense graphs. On the other hand, clique-width is not closed under taking subgraphs (or minors), just induced subgraphs.

The tree-depth of a graph has been defined in [16], and is equivalent or similar to notions such as the *vertex ranking number* and the minimum height of an *elimination tree* [1,5,20], etc. Graphs with bounded tree-depth are sparse, and enjoy strong "finiteness" properties (finiteness of cores, existence of non-trivial

* Supported by the research centre Institute for Theoretical Computer Science (CE-ITI); Czech Science foundation project No. P202/12/G061.

** Supported by the project LL1201 CORES of the Ministery of Education of Czech republic.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 419–430, 2012.

automorphism if the graph is large, well quasi-ordering by subgraph inclusion order). They received almost immediate attention and play a central role in the theory of graph classes of bounded expansion [17].

Graphs of bounded parameters such as clique-width allow us to efficiently solve various optimization problems which are difficult (e.g. NP-hard) in general [3,6,12,11]. However, instead of solving each problem separately, we may ask for results which give a solution to a whole class of problems. We call such results *algorithmic metatheorems.* One of the most famous results of this kind is Courcelle's theorem [2], which states that every graph property expressible in MSO_2 logic of graphs can be solved in linear time on graphs of bounded tree-width. More precisely, the MSO_2 model-checking problem for a graph G of tree-width $\mathrm{tw}(G)$ and a formula ϕ, i.e. the question whether $G \models \phi$, can be solved in time $\mathcal{O}(|G| \cdot f(\phi, \mathrm{tw}(G)))$. (In the world of parameterized complexity we say that such problems, solvable in time $\mathcal{O}(n^p \cdot f(k))$ for some constant p and a computable function f, where k is some parameter of the input and n the size of the input, are *fixed-parameter tractable (FPT).*) For clique-width a result similar to Courcelle's theorem holds: MSO_1 model checking is FPT on graphs of bounded clique-width [3].

However, an issue with these results is that, as showed by Frick and Grohe [7] for MSO model checking of the class of all trees, the function f of Courcelle's algorithm is, unavoidably, non-elementary in the parameter ϕ (unless P=NP). This brings the following question: Are there any interesting graph classes where the dependency on the formula is better? Only recently, in 2010, Lampis [15] gave an FPT algorithm for MSO_2 model checking on graphs of bounded vertex cover with elementary (doubly-exponential) dependence on the formula. A current result of Gajarský and Hliněný [9] shows that there exists an FPT algorithm for MSO_2 model checking for graphs of bounded tree-depth, again with elementary dependency on the formula.

Our Results. Motivated by the success of tree-depth, we would like to formalize a parameter which extends tree-depth towards a logic-flavoured graph description such as that of clique-width. We start by introducing two such parameters: *shrub-depth* and *SC-depth.* Both of these parameters are based on the notion of *tree-model*, which can be seen as a minimalistic analogue of graph interpretation into a tree. Shrub-depth and SC-depth are then defined in terms of the number of layers (the depth) such a tree-model must have to be able to interpret a given graph.

The first main result of this paper is that the classes of the graphs resulting from an MSO_1 graph interpretation in the class of all finite rooted trees of height $\leq d$, with vertices labelled by a finite set of labels, are exactly the classes of graphs of shrub-depth at most d. This result, in combination with [9], leads to an FPT algorithm for MSO_1 model checking for graphs of bounded shrub-depth (SC-depth) with an elementary dependence on the formula.

Continuing in the same direction we also introduce the notion of m-partite cographs, which are a natural extension of ordinary cographs. (Recall that a *cograph* is a graph that can be generated from K_1 by complementations and disjoint unions.) We argue that m-partite cographs represent a smooth intermediate transition from the shrub- and SC-depth to the significantly wider and established notions of clique-width [4] and NLC-width [21]. Indeed, we show that all graphs in any class of shrub-depth d are m-partite cographs with a representation of depth $\leq d$, for suitable m. On the other hand, every m-partite cograph has clique-width at most $2m$.

The second main result of this paper is that the class of m-partite cographs is well-quasi-ordered by the relation of "is an induced subgraph of". This is a significant result, which implies that a) testing whether a graph is an m-partite cograph is an FPT problem, and b) deciding any hereditary property (i.e. property closed under taking induced subgraphs) on this class is an FPT problem.

Paper Organization. In Section 2 we give the necessary definitions, including $\mathrm{MSO}_1/\mathrm{MSO}_2$ logics and FO/MSO graph interpretation, which is a specialized instance of the concept of interpretability of logic theories. Section 3 then introduces tree models and, through them, the new invariants shrub-depth and the related SC-depth. Section 4 deals with MSO model checking and interpretability. In Section 5 we investigate the more general concept of m-partite cographs and relate it to other invariants. We conclude with Section 6.

2 Definitions

We assume the reader is familiar with standard notation of graph theory. In particular, all our graphs (both directed and undirected) are finite and simple (i.e. without loops or multiple edges). For a graph $G = (V, E)$ we use $V(G)$ to denote its vertex set and $E(G)$ the set of its edges. We will often use *labelled graphs*, where each vertex is assigned one of some fixed finite set of labels. A forest F is a graph without cycles, and a tree T is a forest with a single connected component. We will consider mainly *rooted forests (trees)*, in which every connected component has a designated vertex called the *root*. The *height* of a vertex x in a rooted forest F is the length of a path from the root (of the component of F to which x belongs) to x and is noted $\mathrm{height}(x, F)$. The *height*[1] of the rooted forest F is the maximum height of the vertices of F. Let x, y be vertices of F. The vertex x is an *ancestor* of y, and y is a *descendant* of x, in F if x belongs to the path of F linking y to the corresponding root. If x is an ancestor of y and $xy \in E(T)$, then x is called a *parent* of y, and y is a *child* of x.

Tree-Depth. The *closure* $\mathrm{Clos}(F)$ of a forest F is the graph obtained from F by making every vertex adjacent to all of its ancestors. The *tree-depth* $\mathrm{td}(G)$ of a graph G is one more than the minimum height of a rooted forest F such that

[1] There is a conflict in the literature about whether the height of a rooted tree should be measured by the "root-to-leaves distance" or by the "number of levels" (a difference of 1 on finite trees). We adopt the convention that the height of a single-node tree is 0 (i.e., the former view).

$G \subseteq \mathrm{Clos}(F)$ [16]. For a proof of the following proposition, as well as for a more extensive study of tree-depth, we refer the reader to [18].

Proposition 2.1. *Let G and H be graphs. Then the following is true:*

- *If H is a minor of G then* $\mathrm{td}(H) \leq \mathrm{td}(G)$*.*
- *If L is the length of a longest path in G then* $\lceil \log_2(L+2) \rceil \leq \mathrm{td}(G) \leq L+1$*.*
- *If* $\mathrm{tw}(G)$ *and* $\mathrm{pw}(G)$ *denote in order the tree-width and path-width of a graph, then* $\mathrm{tw}(G) \leq \mathrm{pw}(G) \leq \mathrm{td}(G) - 1$*.*

Clique-Width. A *k-expression* is an algebraic expression with the following four operations on vertex-labelled graphs using k labels: create a new vertex with label i; take the disjoint union of two labelled graphs; add all edges between vertices of label i and label j; and relabel all vertices with label i to have label j. The *clique-width* [4] of a graph G equals the minimum k such that (some labelling of) G is the value of a k-expression.

MSO Logic and Interpretation. We now briefly introduce the *monadic second order logic (*MSO*)* over graphs and the concept of FO (MSO) graph interpretation. MSO is the extension of first-order logic (FO) by quantification over sets:

Definition 2.2 (MSO_1 **logic of graphs).** *The language of* MSO_1 *consists of expressions built from the following elements:*

- *variables $x, y, \ldots$ for vertices, and X, Y for sets of vertices,*
- *the predicates $x \in X$ and* $\mathrm{edge}(x, y)$ *with the standard meaning,*
- *equality for variables, quantifiers $\forall, \exists$ ranging over vertices and vertex sets, and the standard Boolean connectives.*

MSO_1 logic can be used to express many interesting graph properties, such as 3-colourability. We also mention MSO_2 logic, which additionally includes quantification over edge sets and can express properties which are not MSO_1 definable (e.g. Hamiltonicity). The large expressive power of both MSO_1 and MSO_2 makes them a very popular choice when formulating algorithmic metatheorems (e.g., for graphs of bounded clique-width or tree-width, respectively).

A useful tool when solving the model checking problem on a class of structures is the ability to "efficiently translate" an instance of the problem to a different class of structures, for which we already have an efficient model checking algorithm. To this end we introduce simple FO/MSO_1 graph interpretation, which is an instance of the general concept of interpretability of logic theories [19] restricted to simple graphs with vertices represented by singletons.

Definition 2.3. *A* FO (MSO_1) graph interpretation *is a pair $I = (\nu, \mu)$ of* FO *(*MSO_1*) formulae (with* 1 *and* 2 *free variables respectively) in the language of graphs, where μ is symmetric (i.e. $G \models \mu(x, y) \leftrightarrow \mu(y, x)$ in every graph G). To each graph G it associates a graph $I(G)$ (by standard abuse of notation), which is defined as follows:*

- *The vertex set of $I(G)$ is the set of all vertices v of G such that $G \models \nu(v)$;*
- *The edge set of $I(G)$ is the set of all the pairs $\{u, v\}$ of vertices of G such that $G \models \nu(u) \wedge \nu(v) \wedge \mu(u, v)$.*

This definition naturally extends to the case of vertex-labelled graphs (using a finite set of labels, sometimes called colours) by introducing finitely many unary relations in the language to encode the labelling.

3 Capturing Height of Graphs

To motivate the definition of shrub-depth, we recall the simple *neighbourhood diversity* parameter introduced by Lampis [15] in his search for graph classes having a faster MSO_1 model checking algorithm: Two vertices u, v are *twins* in a graph G if $N_G(u) \setminus \{v\} = N_G(v) \setminus \{u\}$. The *neighbourhood diversity* of a graph G is the smallest m such that $V(G)$ can be partitioned into m sets such that in each part the vertices are pairwise twins (each part is then either a clique or independent). This basically means that $V(G)$ can be coloured by m labels such that the existence of an edge uv depends solely on the labels of u and v.

Inspired by some subsequent generalizations of neighbourhood diversity, e.g, in [10,8], our idea is to enrich it with a bounded number of "layers". That is, we bring the following definition, which can also be viewed as a very simplified (or minimalistic) analogue of a graph interpretation (Def. 2.3) into a tree of bounded height:

Definition 3.1 (Tree-model). *We say that a graph G has a* tree-model of m labels and depth d *if there exists a rooted tree T (of height d) such that*

i. the set of leaves of T is exactly $V(G)$,
ii. the length of each root-to-leaf path in T is exactly d,
*iii. each leaf of T is assigned one of m labels (T is m-*labelled*),*
iv. and the existence of a G-edge between $u, v \in V(G)$ depends solely on the labels of u, v and the distance between u, v in T.

The class of all graphs having a tree-model of m labels and depth d is denoted by $\mathcal{TM}_m(d)$.

Note that there is no explicit computability assumption in Definition 3.1.iv; it is implicit from the fact that a tree-model has fixed height and uses a bounded number of labels.

For instance, $K_n \in \mathcal{TM}_1(1)$ or $K_{n,n} \in \mathcal{TM}_2(1)$. Definition 3.1 is further illustrated in Figure 1. It is easy to see that each class $\mathcal{TM}_m(d)$ is closed under complements and induced subgraphs, but neither under disjoint unions, nor under subgraphs. The depth of a tree model generalizes tree-depth of a graph as follows (while the other direction is obviously unbounded, e.g., for cliques):

Proposition 3.2. *If G is of tree-depth d, then $G \in \mathcal{TM}_{2^d}(d)$, and $G \in \mathcal{TM}_{2^d}(d-1)$ if, moreover, G is connected.*

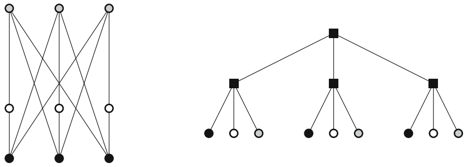


Fig. 1. The graph obtained from $K_{3,3}$ by subdividing a matching belongs to $\mathcal{TM}_3(2)$

In the technical definition of a tree-model, the depth parameter d is much more important (for our purposes, e.g. efficient MSO property testing) than the number of labels m. With this in mind, it is useful to work with a more streamlined notion which only requires a single parameter, and to this end we introduce the following:

Definition 3.3 (Shrub-depth). *A class of graphs $\mathcal{G}$ has* shrub-depth *d if there exists m such that $\mathcal{G} \subseteq \mathcal{TM}_m(d)$, while for all natural m it is $\mathcal{G} \not\subseteq \mathcal{TM}_m(d-1)$.*

Note that Definition 3.3 is asymptotic as it makes sense only for infinite graph classes; the shrub-depth of a single finite graph is always at most one (0 for empty or one-vertex graphs). Furthermore, it makes no sense to say "the class of all graphs of shrub-depth d".

For instance, the class of all cliques has shrub-depth 1. For more relations of shrub-depth to other established concepts such as cographs or clique-width we refer the reader to Section 5. It is, however, immediate from Definition 3.1 that all graphs in $\mathcal{TM}_m(d)$ have clique-width $\leq m$, and our bounded shrub-depth indeed "lies between" bounded tree-depth and bounded clique-width:

Proposition 3.4. *Let $\mathcal{G}$ be a graph class and d an integer. Then:*

a) If $\mathcal{G}$ is of tree-depth $\leq d$, then $\mathcal{G}$ is of shrub-depth $\leq d$ (cf. Proposition 3.2).
b) If $\mathcal{G}$ is of bounded shrub-depth, then $\mathcal{G}$ is of bounded clique-width.

The converse statements are not true in general.

SC-Depth. One can come with yet another, very simple and single-parameter based, definition of a depth-like parameter which is asymptotically equivalent to shrub-depth: Let G be a graph and let $X \subseteq V(G)$. We denote by $\overline{G}^X$ the graph G' with vertex set $V(G)$ where $x \neq y$ are adjacent in G' if (i) either $\{x,y\} \in E(G)$ and $\{x,y\} \not\subseteq X$, or (ii) $\{x,y\} \notin E(G)$ and $\{x,y\} \subseteq X$. In other words, $\overline{G}^X$ is the graph obtained from G by complementing the edges on X.

Definition 3.5 (SC-depth[2]). *We define inductively the class $\mathcal{SC}(n)$ as follows:*

- *We let $\mathcal{SC}(0) = \{K_1\}$;*
- *if $G_1, \ldots, G_p \in \mathcal{SC}(n)$ and $H = G_1 \dot\cup \ldots \dot\cup G_p$ denotes the disjoint union of the G_i, then for every subset X of vertices of H we have $\overline{H}^X \in \mathcal{SC}(n+1)$.*

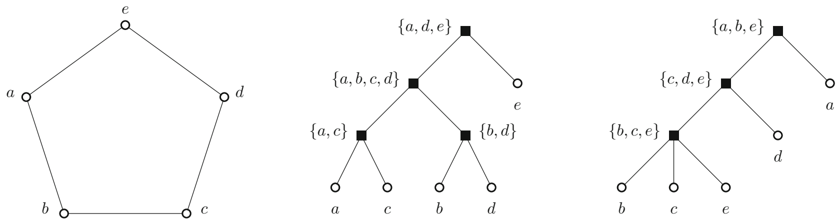


Fig. 2. A graph G and two possible SC-depth representations by depicted trees

The SC-depth *of* G *is the minimum integer* n *such that* $G \in \mathcal{SC}(n)$.

The SC-depth of a graph G is thus the minimum height of a rooted tree Y, such that the leaves of Y form the vertex set of G, and each internal node v is assigned a subset X of the descendant leaves of v. Then the graph corresponding to v in Y is the complement on X of the disjoint union of the graphs corresponding to the children of v (see Fig. 2).

Theorem 3.6. *Let* $\mathcal{G}$ *be a class of graphs. Then the following are equivalent:*

- *There exist integers* d, m *such that* $\mathcal{G} \subseteq \mathcal{TM}_m(d)$ *(i.e.* $\mathcal{G}$ *has bounded shrub-depth).*
- *There exists an integer* k *such that* $\mathcal{G} \subseteq \mathcal{SC}(k)$ *(i.e.* $\mathcal{G}$ *has bounded SC-depth).*

The reason we introduce both asymptotically equivalent SC-depth and shrub-depth measures here is that each one brings a unique perspective on the class of graphs we are interested in (and for a yet another, more general, perspective we refer to Section 5).

4 MSO Interpretation and Model Checking

In this section we present the first main result of the paper, Theorem 4.1, which shows that our tree-model (Def. 3.1) indeed fully captures the power of an MSO$_1$ graph interpretation. While such a result may be expected for FO logic, we believe it is rather surprising in the full scope of MSO logic. The assumption of bounded depth of the target tree is absolutely essential here.

Theorem 4.1. *A class* $\mathcal{G}$ *of graphs has an* MSO$_1$ *graph interpretation in the class of all finite rooted trees of height* $\leq d$*, with vertices labelled by a finite set of labels, if and only if* $\mathcal{G}$ *has shrub-depth at most* d.

While the proof of Theorem 4.1 is rather involved by itself, it strongly relies also on the following recent result which is of independent interest:

[2] As the "Subset-Complementation" depth.

Proposition 4.2 (Gajarský and Hliněný [9]). *Let T be a rooted tree with each vertex assigned one of a finite set of m labels, and let ϕ be any* MSO$_1$ *sentence with q quantifiers. There exists a function*[3] $R(q, m, d) \leq exp^{(d)}\big((q + m)^{\mathcal{O}(1)}\big)$ *such that the following holds:*

Assume a node $u \in V(T)$ such that the subtree $T_u \subseteq T$ rooted at u is of height d, and denote by $U_1, U_2, \ldots, U_k$ the connected components of $T_u - u$. If there is $I \subseteq \{2, \ldots, k\}$, $|I| \geq R(q, m, d)$, such that there exist label-preserving isomorphisms from U_1 to each U_i, $i \in I$, let $T' = T - V(U_1)$. Then, $T \models \phi \iff T' \models \phi$.

Theorem 4.1, combined with another recent result of [9] — stated here as Theorem 4.3, gives us an MSO$_1$ model checking algorithm which is in FPT and has an elementary dependence on the size of the formula ϕ. Note that this claim is in contrast with the well known algorithms of Courcelle [2] for MSO$_2$ and Courcelle et al. [3] for MSO$_1$ model checking problems on the graphs of bounded tree- and clique-width, both of which were shown to have a non-elementary lower bound in ϕ by Frick and Grohe [7].

Theorem 4.3 (Gajarský and Hliněný [9]). *Assume $d \geq 1$ is a fixed integer. Let T be a rooted tree of height d with vertices labelled by a finite set of m labels, and let ϕ be any* MSO$_1$ *sentence with q quantifiers. Then the* MSO$_1$ *model checking problem $T \models \phi$ can be solved by an FPT algorithm, concretely in time $\mathcal{O}\big(|V(T)|\big) + exp^{(d+1)}\big((q + m)^{\mathcal{O}(1)}\big)$, which is elementary in the parameters ϕ and m.*

Corollary 4.4. *Assume $d \geq 1$ is a fixed integer. Let $\mathcal{G}$ be any graph class of shrub-depth (or SC-depth) $\leq d$. Then the* MSO$_1$ *model checking problem on $\mathcal{G}$, i.e., testing $G \models \phi$ for the input $G \in \mathcal{G}$ and* MSO$_1$ *sentence ϕ, can be solved by an FPT algorithm, the runtime of which has an elementary dependence on the parameter ϕ. This assumes G is given on the input alongside with its tree-model of depth d.*

Corollary 4.4 thus nicely complements Theorem 4.3 and its straightforward consequence in fast MSO$_2$ model checking of graphs of bounded tree-depth. The converse direction of Theorem 4.1 moreover shows that bounded shrub-depth exactly characterizes the largest extent to which faster MSO$_1$ model checking can be obtained by the means of Theorem 4.3 – the primary motivation of our research here.

Note that the result of Corollary 4.4 assumes we are given a tree-model of G of depth d on the input. It is therefore natural to ask what is the complexity of obtaining such a model. So far, we have not reached much progress in this direction—while we can test in FPT whether a graph belongs to $\mathcal{TM}_m(d)$ (Section 5), we are not yet able to construct a corresponding tree-model (or, alternatively, an SC-depth tree).

[3] Here $exp^{(d)}$ stands for the iterated ("tower of height d") exponential, i.e., $exp^{(1)}(x) = 2^x$ and $exp^{(i+1)}(x) = 2^{exp^{(i)}(x)}$.

5 On m-Partite Cographs

A *cograph*, or *complement-reducible* graph, is a graph that can be generated from K_1 by complementations and disjoint unions. Cographs were introduced independently by several authors in the seventies, and they are exactly those graphs excluding an induced path of length three. The tree representation of a cograph G is a rooted tree T (called *cotree*), whose leaves are the vertices of G and whose internal nodes represent complemented union.

Some generalizations of cographs have been proposed; e.g., bi-cographs [13] or k-cographs [14]. The following generalization we present here is very natural:

Definition 5.1 (m-partite cograph). *An m-*partite cograph *is a graph that admits an m-*partite cotree *representation, that is a rooted tree T such that*

- *the leaves of T are the vertices of G, and are coloured by a label from $\{1, \dots, m\}$,*
- *the internal nodes v of T are assigned symmetric functions $f_v : \{1, \dots, m\} \times \{1, \dots, m\} \to \{0, 1\}$ with the property that two vertices x and y of G with respective colours i and j are adjacent iff their least common ancestor v in T has $f_v(i, j) = 1$.*

By extension, the depth *of an m-partite cograph G is the minimum height of an m-partite cotree representation of G.*

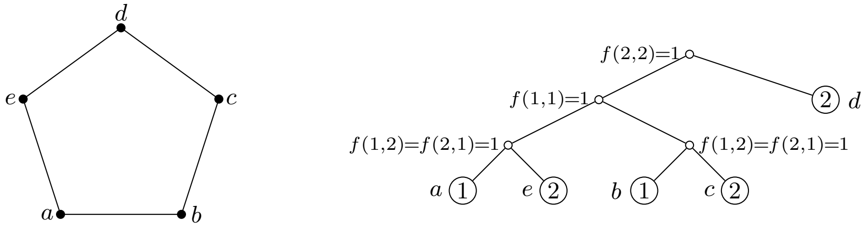


Fig. 3. A 2-partite cotree representation of the graph cycle C_5, with $f(x, y) = 0$ unless otherwise specified

One can easily deduce from the definition that the graphs in $\mathcal{TM}_m(d)$ are all m-partite cographs of depth $\leq d$. A converse claim is also true (although not immediate).

Theorem 5.2. *Let $\mathcal{G}$ be a graph class. Then the following are equivalent:*

- *There exist integers d, m such that $\mathcal{G}$ only contains m-partite cographs of depth $\leq d$.*
- *The class $\mathcal{G}$ has bounded shrub-depth (or bounded SC-depth, cf. Theorem 3.6).*

It is instructive to look at the general relation of m-partite cographs to shrub-depth and clique-width. The crucial difference between a tree-model and an m-partite cotree representation is in bounding the height of the former. Comparing an m-partite cotree representation to a clique-width expression, roughly saying, the difference lies in the absence of the relabelling operator for the former (this is better seen with the related NLC-width notion [21]). Altogether we get:

Proposition 5.3. *Every m-partite cograph has clique-width at most $2m$.*

Figure 4 summarizes the inclusion hierarchy of the classes we have considered. We give next two examples illustrating the fact that the inclusions indicated in the figure are strict.

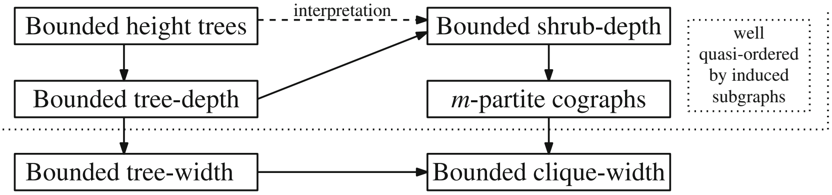


Fig. 4. Hierarchy of graph classes. Arrows mean strict inclusion.

Example 5.4. a) Let H_n denote the graph obtained from the disjoint union of an independent set $\{a_1, \ldots, a_n\}$ and a clique on $\{b_1, \ldots, b_n\}$ by adding all edges a_ib_j such that $i \geq j$. Although each H_n is a 2-partite cograph, the class $\{H_n\}$ has unbounded shrub-depth.

b) The class of all paths has clique-width ≤ 3, while a path of length n is an m-partite cograph if and only if $n < 3(2^m - 1)$.

A *well-quasi-ordering* (or *wqo*) of a set X is a quasi-ordering such that for any infinite sequence of elements $x_1, x_2, \ldots$ of X there exist $i < j$ with $x_i \leq x_j$. In other words, a wqo is a quasi-ordering that does not contain an infinite strictly decreasing sequence or an infinite set of non-comparable elements (i.e. an infinite *antichain*). The existence of a well-quasi-ordering has great algorithmic consequences. For instance, the Robertson-Seymour theorem, which proves that the relation “is a minor of” is a well-quasi-ordering of graphs, implies that for every minor-closed family $\mathcal{C}$ there is a finite set of forbidden minors for $\mathcal{C}$, and hence there is a polynomial time algorithm for testing whether a graph belongs to $\mathcal{C}$. Here we will focus on the quasi-ordering $\subseteq_i$ (“is an induced subgraph of”). A class $\mathcal{C}$ that is closed under taking induced subgraphs is said to be *hereditary.*

Theorem 5.5. *Let m be an integer. The class of m-partite cographs is well-quasi-ordered by the relation $\subseteq_i$ (“is an induced subgraph of”).*

Corollary 5.6. *a) For every integer m, the class of m-partite cographs is defined by a finite set of excluded induced subgraphs. Hence m-partite cographs are recognizable by an FPT algorithm.*

b) For every hereditary property $\mathcal{P}$ and every integer m, the property $\mathcal{P}$ can be decided by an FPT algorithm on the class of m-partite cographs.

For instance, for every fixed integers k and m, the k-colourability problem can be solved in polynomial time in the class of m-partite cographs.

6 Concluding Notes

The main motivation of this paper has been to come up with a notion of "bounded graph depth" which extends the established notion of tree-depth towards dense graphs and parameters similar to clique-width. We have succeeded in this direction with two new, asymptotically equivalent, parameters; shrub-depth and SC-depth. The advantage of the former is that it exactly characterizes the graph classes interpretable in trees of height d, while the latter (SC-depth) outdoes the former with a simpler, single-parameter definition.

Our research topic is also closely related to the class of cographs, and to their natural generalization — m-partite cographs. Saying briefly, graphs of bounded SC-depth are those having m-partite cograph representation of bounded depth. On the other hand, the larger class of all m-partite cographs is "sandwiched" strictly between bounded shrub-depth and bounded clique-width, and it shares several nice finiteness properties with classes of bounded shrub-depth (e.g., well-quasi-ordering under induced subgraphs, which is not true for bounded clique-width classes in general).

The prime algorithmic applications are of two kinds: Firstly, in connection with [9], we obtained an FPT algorithm for MSO_1 model checking for graph classes of bounded shrub-depth which is faster than the algorithm of [3] in the sense that it depends on the checked formula in an elementary way. Secondly, via the well-quasi-ordering property of m-partite cographs, we have proved another algorithmic metatheorem claiming FPT (nonuniform) decidability of all hereditary properties on the classes of m-partite cographs.

Finally, we would like to mention some open questions and directions for future research. Primarily, we do not know yet how to efficiently (in FPT) construct decompositions related to our depth parameters (in this respect our situation is similar to that of clique-width), though we can test existence of such decompositions via Corollary 5.6. We also suggest to investigate the complexity of the isomorphism (or canonical labelling) problem on the classes of bounded shrub-depth, and to try to characterize the maximal graph classes admitting well-quasi-ordering under coloured induced subgraphs. Finally, we remark on the possibility of extending our tools and notions from graphs to general relational structures.

References

1. Bodlaender, H., Deogun, J., Jansen, K., Kloks, T., Kratsch, D., Müller, H., Tuza, Z.: Rankings of Graphs. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 292–304. Springer, Heidelberg (1995)

2. Courcelle, B.: The monadic second order logic of graphs I: Recognizable sets of finite graphs. Inform. and Comput. 85, 12–75 (1990)
3. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory Comput. Syst. 33(2), 125–150 (2000)
4. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Appl. Math. 101(1-3), 77–114 (2000)
5. Deogun, J., Kloks, T., Kratsch, D., Muller, H.: On Vertex Ranking for Permutation and Other Graphs. In: Enjalbert, P., Mayr, E.W., Wagner, K.W. (eds.) STACS 1994. LNCS, vol. 775, pp. 747–758. Springer, Heidelberg (1994)
6. Espelage, W., Gurski, F., Wanke, E.: How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In: Brandstädt, A., Van Le, B. (eds.) WG 2001. LNCS, vol. 2204, pp. 117–128. Springer, Heidelberg (2001)
7. Frick, M., Grohe, M.: The complexity of first-order and monadic second-order logic revisited. Ann. Pure Appl. Logic 130(1-3), 3–31 (2004)
8. Gajarský, J.: Efficient solvability of graph MSO properties. Master's thesis, Masaryk University, Brno (2012)
9. Gajarský, J., Hliněný, P.: Deciding graph MSO properties: Has it all been told already (submitted, 2012)
10. Ganian, R.: Twin-Cover: Beyond Vertex Cover in Parameterized Algorithmics. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 259–271. Springer, Heidelberg (2012)
11. Ganian, R., Hliněný, P., Obdržálek, J.: Clique-width: When hard does not mean impossible. In: STACS 2011. LIPIcs, vol. 9, pp. 404–415. Dagstuhl Publishing (2011)
12. Gerber, M.U., Kobler, D.: Algorithms for vertex-partitioning problems on graphs with fixed clique-width. Theoret. Comput. Sci. 299(1-3), 719–734 (2003)
13. Giakoumakis, V., Vanherpe, J.-M.: Bi-complement reducible graphs. Adv. Appl. Math. 18, 389–402 (1997)
14. Hung, L.-J., Kloks, T.: k-cographs are Kruskalian. Chic. J. Theoret. Comput. Sci. 2011, 1–11 (2011)
15. Lampis, M.: Algorithmic Meta-theorems for Restrictions of Treewidth. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 549–560. Springer, Heidelberg (2010)
16. Nešetřil, J., Ossona de Mendez, P.: Tree-depth, subgraph coloring and homomorphism bounds. European J. Combin. 27(6), 1024–1041 (2006)
17. Nešetřil, J., Ossona de Mendez, P.: Grad and classes with bounded expansion I. Decompositions. European J. Combin. 29(3), 760–776 (2008)
18. Nešetřil, J., Ossona de Mendez, P.: Sparsity (Graphs, Structures, and Algorithms) Algorithms and Combinatorics, vol. 28, p. 465. Springer (2012)
19. Rabin, M.O.: A simple method for undecidability proofs and some applications. In: Bar-Hillel, Y. (ed.) Logic, Methodology and Philosophy of Sciences, vol. 1, pp. 58–68. North-Holland, Amsterdam (1964)
20. Schaffer, P.: Optimal node ranking of trees in linear time. Inform. Process. Lett. 33, 91–96 (1989/1990)
21. Wanke, E.: k-NLC graphs and polynomial algorithms. Discrete Appl. Math. 54, 251–266 (1994)

Strategy Machines and Their Complexity

Marcus Gelderie[*]

RWTH Aachen, Lehrstuhl für Informatik 7,
Logic and Theory of Discrete Systems,
D-52056 Aachen
gelderie@automata.rwth-aachen.de

Abstract. We introduce a machine model for the execution of strategies in (regular) infinite games that refines the standard model of Mealy automata. This model of controllers is formalized in the terminological framework of Turing machines. We show how polynomially sized controllers can be found for Muller and Streett games. We are able to distinguish aspects of executing strategies ("size", "latency", "space consumption") that are not visible in Mealy automata. Also, lower bound results are obtained.

1 Introduction

Strategies obtained from ω-regular games are used to obtain controllers for reactive systems. The controllers obtained in this way are transition systems with output, also called Mealy machines. The physical implementation of such abstract transition systems raises interesting questions, starting with the observation that the naive implementation of a Mealy machine by a physical machine (such as a circuit or a register machine) essentially amounts to encoding a large case distinction based on the current input and state. This approach entails considerable complexity challenges [1], as it essentially preserves the size of the underlying Mealy machine. These machines are known to be large in general [2]. This is reflected in the complexity of solving ω-regular games [3,4,5,6]. Optimization of Mealy machines has been investigated [7,8], but must ultimately obey the general bounds mentioned. These observations indicate that pursuing a direct approach, without the detour via Mealy machines, in synthesizing controllers may be worthwhile. We propose a new model for reasoning about such physical machines and their synthesis. This model, called a *strategy machine*, is based on an appropriate format of Turing machines. The concept of a Turing machine is widely used in theoretical scenarios to model computational systems, such as [9]. A strategy machine is a multi-tape Turing machine with two distinguished tapes, one for input and output and another for storing information from one computation to the next. Referring to a given game graph, the code of an input vertex appears on the IO-tape and an output is produced. The process is then repeated. This model introduces new criteria for evaluating a strategy.

[*] Supported by DFG research training group 1298, "Algorithmic Synthesis of Reactive and Discrete-Continuous Systems" (AlgoSyn).

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 431–442, 2012.

For example, it now makes sense to investigate the number of steps required to transform an input into the corresponding output, called the *latency*. Likewise we may ask how much information needs to be stored from the computation of one output to the computation of the next output. Note that translating a Mealy machine into this model entails the problems discussed above and yields a machine of roughly equal size (the number of control states).

Introducing this model, we present initial results for two classes of ω-regular games, namely Muller and Streett games. Building on Zielonka's algorithm [10], we show that, in both cases, we may construct strategy machines implementing a winning strategy of an exponentially lower size than any Mealy machine winning strategy: Both the size of the strategy machine and the amount of information stored on the tape are bounded polynomially in the size of the arena and of the winning condition (given by a propositional formula ϕ or a set of Streett pairs). Moreover, for Streett games even the latency is bounded polynomially in these parameters. For Muller games the latency is linear in the size of the enumerative representation of the winning condition. We also present lower bounds for the latency and space requirement by translating some well known results from the theory of Mealy machine strategies to our model. This yields lower bounds for Muller, Streett, and LTL games.

In related work, Madhusudan [11] considers the synthesis of reactive programs over Boolean variables. A machine executing such a program falls into the model discussed above. The size of the program then loosely corresponds to the number of control states. In the same way, the requirement of tape cells loosely corresponds to the number of Boolean variables. The latency and space requirement are not studied in [11].

The paper is an extended abstract of [12]. It is structured as follows. First, we give a formal introduction of the Turing machine model mentioned above. We formally define the parameters latency, space requirement, and size. Next, we recall some elementary concepts of the theory of infinite two player zero-sum games with ω-regular winning conditions. We show lower bounds for the latency and space requirement of machines implementing winning strategies in Muller, Streett, and LTL games. Then we develop an adaptive algorithm for Muller games. This algorithm is based on Zielonka's construction [10], using some ideas from [2]. The latency and space consumption strongly depend on the way the winning condition is given. We illustrate this fact by showing how the algorithm can be used to obtain an efficient controller – that is, one with latency, size, and space requirement bounded polynomially in the size of the arena and the winning condition – if the winning condition is a Streett condition. This is promising, because Streett conditions are more succinct than explicit Muller conditions.

2 Strategy Machines – A Formal Model

The intuition of a *strategy*[1] *machine* is that of a "black box" which receives an input (a bit-string), does some internal computation and, at some point,

[1] For a definition of a strategy, the reader may want to skip ahead to the next section.

produces an output. It then receives the next input and so forth. We will refer to such a sequence of steps – receive an input, compute, produce an output – as an *iteration.* In general, it is allowed for such a machine to retain some part of its current internal configuration from one iteration to the next. However, it need not do so. Also, the amount of information (how this is quantified will be discussed shortly) is a priori not subject to any restriction. In particular, a strategy machine may require an ever growing amount of memory, increasing from one iteration to the next, to store this information.

Our model of a strategy machine is a deterministic $(k{+}2)$-tape Turing machine $\mathcal{M}$, $k \in \mathbb{N} = \{1, 2, \ldots\}$. The tapes have the following purpose. The first is a designated *IO-tape*, responsible for input to and output from the machine. A bit-string $w \in \mathbb{B}^*$, $\mathbb{B} = \{0, 1\}$, is the content of the IO-tape at the beginning of an iteration. The machine $\mathcal{M}$ is also in a designated *input-state* at this point. Next, $\mathcal{M}$ performs some computation in order to produce an output. During this computation the remaining $k + 1$ tapes may be used. We first discuss the k *computation tapes.* As the name suggests, these tapes are used – as in any Turing machine – to store all the data needed for the computation of the output. The content of the computation tapes is deleted immediately before a new iteration begins. In particular, they cannot be used to store information from one iteration to the next. Storing such information is the purpose of the *memory tape.* Its content is still available during the next iteration. In order to produce an output, $\mathcal{M}$ will write this output, another bit-string, on the IO-tape. Then it will enter a designated *output-state.* Let $\hat{\mathbb{B}} = \mathbb{B} \uplus \{\#\}$. We define:

Definition 1 (k-tape Strategy Machine). *A* strategy machine *is a deterministic $(k + 2)$-tape Turing machine $\mathcal{M} = (Q, \mathbb{B}, \hat{\mathbb{B}}, q_I, q_O, \delta)$ with two designated states q_I and q_O, called the* input-state *and the* output-state *of $\mathcal{M}$ respectively. We require the partial function $\delta\colon Q \times \hat{\mathbb{B}}^{k+2} \dashrightarrow Q \times (\hat{\mathbb{B}} \times \{\leftarrow, \downarrow, \rightarrow\})^{k+2}$ to be undefined for all pairs $(q_O, b_1, \ldots, b_{k+2}) \in Q \times \hat{\mathbb{B}}^{k+2}$. By definition, no transition leads into q_I. The tapes of $\mathcal{M}$ are the* input-output-tape (IO-tape) t_{IO}*, the* computation-tapes $t_{com}^{(i)}$*, $1 \leq i \leq k$, and the* memory-tape t_{mem}.

Hereafter we assume $k = 1$ to simplify the notation. If $k = 1$, we simply write t_{com} for $t_{com}^{(1)}$. Given a strategy machine $\mathcal{M}$, we denote its *size* by $\|\mathcal{M}\| = |Q| - 2$. The size is the number of states, not counting the input and output state. A *configuration* c of $\mathcal{M}$ comprises the current state $q(c)$, the contents of the IO-, computation- and memory-tapes, $t_{IO}(c)$, $t_{com}(c)$, and $t_{mem}(c)$, as well as the current head positions $h_{IO}(c)$, $h_{com}(c)$, and $h_{mem}(c)$ on the respective tapes. It is defined as a tuple $(q(c), t_{IO}(c), t_{com}(c), t_{mem}(c), h_{IO}(c), h_{com}(c), h_{mem}(c)) \in Q \times (\hat{\mathbb{B}}^*)^3 \times \mathbb{Z}^3$. The successor relation on configurations is defined as usual and denoted by $\vdash$. Its transitive closure is denoted by $\vdash^*$. An *iteration* of $\mathcal{M}$ is a sequence $c_1, \ldots, c_l$ of configurations, such that for $1 \leq i \leq l - 1$ we have $c_i \vdash c_{i+1}$. It starts with $c_1 = (q_I, x, \epsilon, w_{mem}, 0, 0, 0)$ and ends with $c_l = (q_O, y, w, w'_{mem}, h, h', h'')$ for some elements $x, y \in \mathbb{B}^*$, arbitrary words $w, w_{mem}, w'_{mem} \in \hat{\mathbb{B}}^*$ and integers $h, h', h'' \in \mathbb{Z}$. We denote iterations by pairs of configurations (c, c'), where the state in c is q_I and that in c' is q_O. Since $\mathcal{M}$

is deterministic, there exists at most one iteration from c to c' (since q_O has no outgoing transitions and q_I has no incoming transitions). If no such iteration exists, the pair (c, c') is an *illegal iteration*. Otherwise, it is a *legal iteration*.

Given a word $u = x_1 \cdots x_n \in A^*$, where $A = \mathbb{B}^*$, a *run* of $\mathcal{M}$ on u is a sequence of legal iterations $(c_1, c_1'), \ldots, (c_n, c_n')$, such that $t_{IO}(c_i) = x_i$ and $t_{mem}(c_i') = t_{mem}(c_{i+1})$. By convention, $t_{mem}(c_1) = \epsilon$. Note that, by the definition of an iteration, $t_{com}(c_i) = \varepsilon$ and $h_{com}(c_i) = h_{IO}(c_i) = h_{mem}(c_i) = 0$. A strategy machine $\mathcal{M}$ defines a function $f\colon A \to A$, where $f(x_1) = t_{IO}(c_1')$. By extension it defines a function $f_{\mathcal{M}}\colon A^*A \to A$, the *function implemented by* $\mathcal{M}$.

The *latency* $T(c, c')$ of a legal iteration (c, c') is the number of configurations on the unique path from c to c'. If the latency of all iterations in any run is bounded by a constant, we define the latency $T(\mathcal{M})$ of $\mathcal{M}$ to be the maximal latency over all such (legal) iterations. Finally, we define the *space requirement* $S(c, c')$ of a legal iteration (c, c') to be the number of tape cells of t_{mem}, visited during that iteration. Again, the space consumption $S(\mathcal{M})$ of $\mathcal{M}$ is the maximum over all space consumptions, if such a maximum exists. Note that both latency and space consumption refer to quantities needed to execute a single iteration, between reading $a \in A$ and outputting $b \in A$.

A Mealy machine is a tuple $\mathfrak{M} = (M, \Sigma, m_0, \delta, \tau)$ with *states* M, *IO-alphabet* Σ, *initial state* m_0, *transition function* $\delta\colon M \times \Sigma \to M$ and *output function* $\tau\colon M \times \Sigma \to \Sigma$. By encoding Σ as a subset of $\mathbb{B}^*$ and maintaining a table of triples $(m, x, \delta(m, x))$ and $(m, x, \tau(m, x))$ in the state space, one obtains a strategy machine $\mathcal{M}$ equivalent to $\mathfrak{M}$:

Proposition 1 (Straightforward Simulation). *For every Mealy machine* $\mathfrak{M} = (M, \Sigma, m_0, \delta, \tau)$ *there exists an equivalent 1-tape strategy machine* $\mathcal{M}_{\mathfrak{M}}$ *of size* $\|\mathcal{M}_{\mathfrak{M}}\| \in \mathcal{O}(|M| \cdot |\Sigma|)$*, space requirement* $S(\mathcal{M}_{\mathfrak{M}}) \in \mathcal{O}(\log_2(|M|))$*, and latency* $T(\mathcal{M}_{\mathfrak{M}}) \in \mathcal{O}(\log_2(|M| \cdot |\Sigma|))$*.*

This illustrates the relationship between a strategy machine and a straightforward simulation of a Mealy machine by a Turing machine: The latter can be seen as a special case of the former. Note also that the complexity hidden in the size of Σ is exposed by this simulation.

3 Basics on Games

We fix the notation and terminology on ω-regular games. We assume the reader is familiar with these concepts. An introduction can be found in e.g. [13,14,15].

An *infinite two player game* (in this paper simply called a *game*) is a tuple $\mathbf{G} = (\mathcal{A}, \varphi)$ with an arena $\mathcal{A} = (V, E)$ and a *winning condition* $\varphi \subseteq V^\omega$. An *arena* is a directed graph $\mathcal{A}$ with the property that every vertex has an outgoing edge. We assume that there is a partition $V = V_0 \uplus V_1$ of the vertex set. There are two players, called player 0 and player 1. Given an initial vertex $v_0 \in V$, they proceed as follows. If $v_0 \in V_0$, player 0 chooses a vertex v_1 in the *neighborhood* vE of v. Otherwise, $v_0 \in V_1$ and player 1 chooses a neighbor v_1. The play then proceeds in the same fashion from the new vertex v_1. In this way the two players create an infinite sequence $\pi = \pi(0)\pi(1)\cdots = v_0v_1\cdots \in V^\omega$ of adjacent vertices $(\pi(i), \pi(i+1)) \in E$, called a *play*. Player 0 wins the play π if $\pi \in \varphi$. Otherwise, player 1 wins. $\mathbf{G}$ is called *ω-regular* if φ is an ω-regular set. All games considered in this paper are ω-regular. A *strategy* for player i is a mapping $\sigma\colon V^*V_i \to V$ assigning a neighbor of v to each string $w \in V^+$ with $\mathrm{last}(w) = v \in V_i$ (where $\mathrm{last}(\cdot)$ denotes the last element of a sequence). π is *consistent* with σ if for every $n \in \mathbb{N}_0$ with $\pi(n) \in V_i$ we have $\pi(i+1) = \sigma(\pi(0)\cdots\pi(n))$. σ is a *winning strategy* for player i if every play consistent with π is won by player i. The *winning region* of player i, written $\mathcal{W}_i$, is the set of vertices $v \in V$, such that player i has a winning strategy σ_v from v. It can be shown that in ω-regular games $V = \mathcal{W}_0 \uplus \mathcal{W}_1$ [16]. If $v \in \mathcal{W}_i$, we say *player i wins from v*. A subarena is an induced subgraph $(V', E \cap V' \times V')$ which is again an arena. An i-trap is a subarena from which player i cannot escape. The *i-attractor on $S \subseteq V$* is the set of vertices from which player i can enforce to visit S and is denoted by $\mathrm{Attr}_i^{\mathcal{A}}(S)$. A corresponding strategy is called an *attractor strategy* (see [13,14]). The complement of an i-attractor is a $(1-i)$-trap. We will need to use the fact that a Turing machine can compute an attractor on S in time polynomial in $|V|$ and $|E|$ (for details see [12]). In this paper we are concerned with only three kinds of winning conditions: Muller, Streett, and LTL conditions. A *Muller condition* is given by a propositional formula ϕ, using the set V as variables. A play $\pi \in V^\omega$ is won by player 0 if the *infinity set* $\mathrm{Inf}(\pi)$ of vertices seen infinitely often in π is a model of ϕ. The equivalent *explicit condition* is $\mathfrak{F} = \{F \subseteq V \mid F \models \phi\}$. It may be exponentially larger than $\|\phi\|$. A game $\mathbf{G} = (\mathcal{A}, \mathfrak{F})$ with a Muller condition $\mathfrak{F}$ is called a *Muller game*. A *Streett condition* is a set $\Omega = \{(R_1, G_1), \ldots, (R_k, G_k)\}$ of pairs of sets $R_i, G_i \subseteq V$. A set $X \subseteq V$ *violates* a Streett pair $(R, G) \in \Omega$ if $R \cap X \neq \emptyset$ but $G \cap X = \emptyset$. A play π *violates* (R, G) if $\mathrm{Inf}(\pi)$ violates (R, G). If π does not violate any pair $(R, G) \in \Omega$, then π *satisfies* Ω and is won by player 0. Otherwise, player 1 wins. A game $\mathbf{G} = (\mathcal{A}, \Omega)$ with a Streett condition Ω is called a *Streett game*. Finally, an *LTL condition* is one where the set of winning plays for player 0, φ, is given by an LTL-formula[2]. If ϕ is an LTL-formula over the propositions V, a play π is won by player 0 iff $\pi \models \phi$. A game with an LTL condition is called an *LTL game*.

4 Lower Bounds

We investigate lower bounds on the space requirement and latency of any machine implementing a winning strategy for player 0 in certain classes of games. Proofs are omitted in the present abstract, but can be found in [12]. Let $f\colon \mathbb{N} \to \mathbb{N}$. Let $(\mathbf{G}_n)_{n\geq 0}$ be a family of games with $\mathbf{G}_n = ((V_n, E_n), \varphi_n)$ and $|V_n| \in \boldsymbol{\mathcal{O}}(n)$, such that no Mealy machine with less than $f(n)$ states implements a winning strategy for player 0 in $\mathbf{G}_n$. Then we say the family $(\mathbf{G}_n)_{n\geq 0}$ is *f-hard*. A class $\mathcal{C}$ of games is *f-hard* if a *witnessing* f-hard family $(\mathbf{G}_n)_{n\geq 0} \subseteq \mathcal{C}$ exists. In general, this definition allows φ_n to be arbitrarily large. However:

[2] Introducing LTL is beyond the scope of this paper (see [13,14]).

Proposition 2. *The classes of Muller and Streett games are 2^n-hard. The conditions (propositional formulas, set of Streett pairs) of the witnessing families are no larger than $\mathcal{O}(n)$. The class of LTL-games has complexity 2^{2^n} with winning condition in the witnessing family of size $\mathcal{O}(n^2)$.*

A *solution scheme for* $\mathcal{C}$ is a mapping $\mathcal{S}$ assigning a strategy machine $\mathcal{S}(\mathbf{G})$ to every $\mathbf{G} \in \mathcal{C}$ which implements a winning strategy for player 0 in $\mathbf{G}$. A function $f \colon \mathbb{N} \to \mathbb{N}$ is *sub-linear* if $f \in \mathcal{O}(x^q)$ for some $q \in (0,1)$. It is well known that, for every $k \in \mathbb{N}$ and every $q \in (0,1)$, one has $\log(x)^k \in \mathcal{O}(x^q)$. Likewise a *sub-exponential function* is a mapping $f \colon \mathbb{N} \to \mathbb{N}$ for which $f \in \mathcal{O}((2^x)^q) = \mathcal{O}(2^{qx})$ for some $q \in (0,1)$. All polynomial functions are sub-exponential. For a proof of the following theorem, see [12]:

Theorem 1 (Lower Bounds)

- *There is no solution scheme $\mathcal{S}$ for the class of Muller games or Streett games which assigns a strategy machine $\mathcal{S}(\mathbf{G})$ to $\mathbf{G} = ((V,E),\varphi)$ that has latency or space requirement sub-linear in $|V|$. In particular, for no $k \in \mathbb{N}$ can there be such a scheme with latency or space requirement in $\log_2(|V|)^k$.*
- *There is no solution scheme for LTL-games, assigning a strategy machine with latency or space requirement sub-exponential in $|V|$ to every LTL-game $\mathbf{G} = ((V,E),\varphi)$. In particular, there can be no such scheme with latency or space requirement polynomial in $|V|$.*

5 Muller and Streett Games

In this section we outline (for formal proofs see [12]) how to construct a polynomial sized strategy machine implementing a winning strategy in Muller and Streett games. For Streett games we even get a machine with a latency polynomial in the winning condition. We will consider a Muller game $\mathbf{G} = (\mathcal{A}, \phi)$, $\mathcal{A} = (V,E)$, with a propositional formula ϕ and explicit condition $\mathfrak{F}$.

Our construction is based on Zielonka's algorithm [10], while also using ideas from [2]. We will define some notation, which will be used subsequently. If $V \in \mathfrak{F}$, we define the set $\max(\mathfrak{F}) = \{V' \subseteq V \mid V' \notin \mathfrak{F} \wedge \forall V'' \supsetneq V' : \ V'' \in \mathfrak{F}\}$ of all maximal subsets of V *not* in $\mathfrak{F}$. If $X \subseteq V$, let $\mathfrak{F} \upharpoonright X := \mathfrak{F} \cap \mathscr{P}(X)$ be the *restriction* of $\mathfrak{F}$ to subsets contained in X. The *Zielonka tree* $\mathcal{Z} := \mathcal{Z}_{\mathfrak{F}} = (N, \lambda)$ of $\mathfrak{F}$ is a labeled tree $N \subseteq \mathbb{N}^*$ with labels $\lambda(x) \in \mathscr{P}(V)$ for all $x \in N$. ε is the root, 1 is the first child of the root, 12 is the second child of 1, and so forth[3]. The root has label $\lambda(\epsilon) = V$. Let $x \in N$ with label $X = \lambda(x)$. If $X \in \mathfrak{F}$ then x has $k = |\max(\mathfrak{F} \upharpoonright X)|$ children $c_0, \ldots, c_{k-1}$, each labeled with a distinct element from $\max(\mathfrak{F} \upharpoonright X)$. If $X \notin \mathfrak{F}$, we apply the same construction with respect to $\mathfrak{F}^{\mathscr{C}}$. The tree obtained in this way is of depth $\leq |V|$. A node labeled with a set $X \in \mathfrak{F}$ is called a 0-level node. The remaining nodes are 1-level nodes.

We will use an additional labeling function κ, which assigns (possibly empty) subarenas to the nodes in N. The idea of attaching the subarenas to the nodes

[3] We assume that, if $n \cdot a \in N$, then $n \cdot b \in N$ for all $1 \leq b \leq a$.

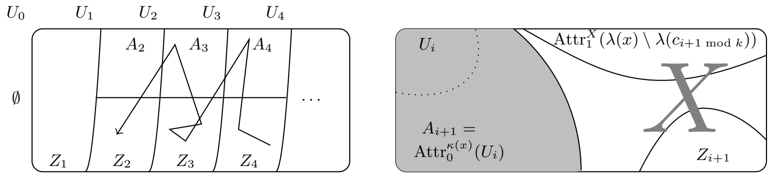


Fig. 1. A play visiting U_i stays in Z_i or is "pulled" into U_{i-1}

in $\mathcal{Z}$ is elaborated in [2]. Given $x \in N$, $\kappa(x)$ is a subarena with the property that player 0 can win the subgame $(\kappa(x), \mathfrak{F} \restriction \lambda(x))$ from every vertex in $\kappa(x)$. The computation of κ is quite involved and only sketched here (see [2]). If x is 0-level with label $\kappa(x)$ and if c is a child of x in $\mathcal{Z}$, the construction is an attractor computation $\kappa(c) = \kappa(x) \setminus \mathrm{Attr}_0^{\kappa(x)}(\lambda(x) \setminus \lambda(c))$. If x is 1-level, one constructs an increasing sequence $(U_i^x)_{i \geq 0}$ of sets of "finished vertices" with $U_0^x = \emptyset$. Let $c_0, \ldots, c_{k-1}$ be the children of x in $\mathcal{Z}$. To compute U_{i+1}^x we remove $A_{i+1} = \mathrm{Attr}_0^{\kappa(x)}(U_i)$ from $\kappa(x)$ and obtain X. Then we remove $\mathrm{Attr}_1^X(\lambda(x) \setminus \lambda(c_{i+1 \bmod k}))$ obtaining Y. Now Z_{i+1}^x is taken to be the winning region in the subgame $(Y, \mathfrak{F} \restriction \lambda(c_{i+1 \bmod k}))$. Then $U_{i+1}^x = A_{i+1} \uplus Z_{i+1}^x$. The label of c_i is $\kappa(c_i) = \bigcup_{j \equiv i \pmod k} Z_j^x$. The intuition is that, once a play reaches U_i^x for some i, it must either stay in Z_i^x or will be forced into U_{i-1}^x. This can only happen finitely many times, so the play will eventually stay in one $Z_i^x \subseteq \lambda(c_{i \bmod k})$ (i.e. in the subtree below $c_{i \bmod k}$) or it must leave the entire subarena $\kappa(x)$ (i.e. the subtree under x). The situation is depicted in Fig. 1.

Lemma 1 ([10,2]). *In the above notation, if $\mathcal{W}_0 = V$ then $\kappa(x) = \bigcup_i U_i^x$. Furthermore, player 0 wins from every $v \in Z_i^x$ in the subgame $(Z_i^x, \mathfrak{F} \restriction \lambda(c_{i \bmod k}))$.*

The sequence $(U_i^x)_{i \geq 1}$ becomes stable after $\leq k \cdot |V|$ steps. We use the superscript x to indicate the (1-level) node for which the sequence has been computed.

We will now describe an *adaptive* strategy machine $\mathcal{M}$ implementing a winning strategy for Muller games. By "adaptive" we mean that the machine continuously refines the memory state it maintains, until eventually a sufficient amount of information is computed to win the play. As a consequence, we must prevent player 0 from inadvertently leaving $\mathcal{W}_0$ during the adaption phase. Hence, we assume $\mathcal{W}_0 = V$, which can be achieved by preprocessing. Since we are interested in representing a winning strategy succinctly, not in deciding the game, this is no restriction. $\mathcal{M}$ will store paths in $\mathcal{Z}$ on its memory tape. By a path we mean one that begins in the root and ends in a leaf. In addition, $\mathcal{M}$ will store the labels $\lambda(x)$ and $\kappa(x)$ for a given node x on the path. Say the current path is $p = p(0), \ldots, p(m)$. Given an input $v \in V$ the machine will then traverse the path and find the unique lowest node $p(i)$ such that $v \in \kappa(p(i))$. The idea is to then play according to the classical Zielonka strategy: If $p(i)$ is 0-level, $\mathcal{M}$ plays an attractor strategy on the set $\lambda(p(i)) \setminus \lambda(p(i+1))$. Otherwise, it will compute the minimal t with $v \in U_t^{p(i)}$ (which exists by Lem. 1) and either update the memory state (if $v \in Z_t^{p(i)}$ to a path through the $(t \bmod k)$-th child of $p(i)$, where k is the number of children of $p(i)$) or it will attract to $U_{t-1}^{p(i)}$.

Unfortunately, since $\kappa(c) = \bigcup_{i \equiv j \pmod{k}} Z_i^x$ is expensive to compute if c is the j-th child of a 1-level node x, the memory update in this approach is costly. Computing the sequence $(Z_i^x)_{i \geq 1}$ requires solving a subgame for every i. Also, to retain all of this information, either in the control states or on the memory tape, would yield a controller of exponential size (or exponential space requirement) in $|V|$. Therefore the idea is to spread the computation of the sets $(U_i^x)_{i \geq 0}$ and $(Z_i^x)_{i \geq 1}$ across multiple iterations of $\mathcal{M}$. This is achieved using an auxiliary Turing machine $\mathcal{M}_\kappa$, which on input $\lambda(x) \in \mathfrak{F}$ and $\kappa(x)$ computes the number k_x of children of x and $(U_i^x)_{i \geq 0}$, $(Z_i^x)_{i \geq 1}$, or rather a compact representation of the same (see below). We attach configurations c of $\mathcal{M}_\kappa$ to the parent of those 0-level nodes x, for which $\kappa(x)$ is currently being computed. We will store the labels $\kappa(x)$ where they have been computed. For 1-level nodes we also store $(U_i^x)_{i \geq 0}$, $(Z_i^x)_{i \geq 1}$ (compactly represented), once available.

There are two obstacles: First, the path may need to be updated, whereby some sets $\kappa(p(i))$ already computed are lost. In Lem. 2 we show that this is tolerable. The second problem is storing the sequence $(U_i^x)_{i \geq 0}$. It may contain exponentially many sets (recall that it becomes stationary after $\leq k \cdot |V|$ steps, where the number k of children of x may be exponential in $|V|$). In the next paragraph, we give a compact representation of the sequence $(U_i^x)_{i \geq 0}$ which allows us to recompute the sets Z_i^x and U_i^x efficiently for any i.

If $D_i^x = U_i^x \setminus U_{i-1}^x$ then $D_i^x \neq \emptyset$ for at most $|V|$ indices i. Given $(D_i^x)_{i \geq 1}$, one obtains Z_i^x by computing $U_{i-1}^x = \bigcup_{j=1}^{i-1} D_i^x$ and $A = \mathrm{Attr}_0^{\kappa(x)}(U_{i-1}^x)$. Then $Z_i^x = U_i^x \setminus A$. If c is the j-th child out of k_x children of x, then $\kappa(c) = \bigcup_{i \equiv j \pmod{k_x}} Z_i$. Thus, it is sufficient to store the number k_x of children of x as well as $\mathcal{D}^x = \{(D_t^x, \mathsf{bin}(t)) \mid D_t^x \neq \emptyset\}$, where $\mathsf{bin}(t)$ is the binary representation of t, in order to compute Z_i^x and $\kappa(c)$ for any child c of x and any i. This representation requires space in $\mathsf{poly}(|V|)$. Computing Z_i^x and the label $\kappa(c)$ requires time in $\mathsf{poly}(|V|, |E|)$.

The memory tape will contain a labeled path p in $\mathcal{Z}$, written as a sequence of triples $p(i) = (n_i, \lambda(p,i), \theta(p,i))$. The i-th node is $n_1 \cdots n_i \in N \subseteq \mathbb{N}^*$, which, by abuse of notation, is denoted by $p(i)$ as well. $p(0)$ is always the root. By convention, $n_0 = 0$. We let $\lambda(p,i) = \lambda(n_1 \cdots n_i)$. For $\theta(p,i)$ we have three possibilities. $\theta(p,i) = \kappa(p,i)$ if this label has already been computed and if $p(i)$ is 0-level. If $p(i)$ is 1-level with $k_{p(i)}$ children, then either $\theta(p,i) = (\kappa(p,i), \mathcal{D}^{p(i)}, k_{p(i)})$, if the set $\mathcal{D}^{p(i)}$ has been computed, or $\theta(p,i) = (\kappa(p,i), c)$ for some configuration c of $\mathcal{M}_\kappa$, otherwise. Finally, $\theta(p,i) = \emptyset$ if the computation has not proceeded this far down the path (note that invoking $\mathcal{M}_\kappa$ on any node $n_1 \cdots n_i$ requires $\kappa(n_1 \cdots n_{i-1})$). We will refer to p as a *memory path* (see Fig. 2). Storing p requires space in $\mathsf{poly}(|V|, S_{\mathcal{M}_\kappa})$, where $S_{\mathcal{M}_\kappa}$ is the space requirement of $\mathcal{M}_\kappa$.

A node with $\theta(p,i) = \emptyset$ is called *untouched*. One with $\theta(p,i) = (\kappa(p,i), c)$ is called *active* (note that it must be 1-level). Nodes which are neither active nor untouched are *finished* and necessarily have a κ-label. We require that, for any memory path p, either all nodes are finished or there exists precisely one node which is active. Since $\kappa(p,i)$ can only be computed when $\kappa(p,i-1)$ is available,

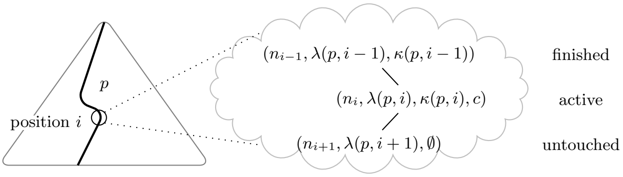

p
position i
$(n_{i-1}, \lambda(p, i-1), \kappa(p, i-1))$
$(n_i, \lambda(p, i), \kappa(p, i), c)$
$(n_{i+1}, \lambda(p, i+1), \emptyset)$
finished
active
untouched

Fig. 2. A Memory Path p

any memory path p with an active node $p(i)$ satisfies that all $p(j)$ with $j < i$ are finished and all $p(r)$ with $r > i$ are untouched.

In addition to the auxiliary machine $\mathcal{M}_\kappa$ we will also make use of an auxiliary machine $\mathcal{M}_\lambda$, which computes the labeling function λ. Both auxiliary machines $\mathcal{M}_\lambda$ and $\mathcal{M}_\kappa$ are used implicitly, as subroutines, by $\mathcal{M}$. Their runtime and space requirement affect the bound on the latency and space requirement of $\mathcal{M}$ we give below[4]. We will assume $\mathcal{M}$ has access to a representation of $\mathcal{A}$ and of the winning condition ϕ in its state space. We do not give the exact construction here (see [12]), but we point out that this requires a number of control states and query time in $\mathsf{poly}(|V|, \|\phi\|)$.

Formally, $\mathcal{M}$ proceeds as follows. The initial memory path p_I is obtained by setting $n_0 = \cdots = n_r = 0$. We choose r, such that $n_1 \cdots n_r \in N$ is a leaf. Let i be minimal with the property that $n_1 \cdots n_i$ is 1-level (i.e. either $i = 0$ or $i = 1$). If $i = 0$, set $p_I(0) = (n_0, \lambda(p_I, 0), (V, c_I))$, where c_I is the initial configuration of $\mathcal{M}_\kappa$. Otherwise, $p_I(0) = (n_0, \lambda(p_I, 0), V)$ and $p_I(1) = (n_1, \lambda(p_I, 1), (\kappa(p_I, 1), c_I))$. The properties of θ described above then imply $p(j) = (n_j, \lambda(p_I, j), \emptyset)$ for all $j > i$. The memory update below will ensure that, for any memory path p, the highest node $p(i)$, such that $\kappa(p, i)$ is not yet available (i.e. $\theta(p, i) = \emptyset$), is 0-level. Note that p_I satisfies this invariant. Additionally, we assume that for any memory path p either all sets $\kappa(p, i)$ are computed or there exists an active node. The memory update will also preserve this invariant.

Suppose the input to $\mathcal{M}$ is $v \in V$ and the current path is p of length $\|p\|$. By assumption, the highest node in p for which κ has not been computed is 0-level. Let i be the maximal index with $\theta(p, i) \neq \emptyset$ and $v \in \kappa(p(i))$, i.e. $\kappa(p, i)$ has been computed and contains v. We call this node the *NOI (node of interest)*. Finding the NOI is in $\mathsf{poly}(|V|)$, as the space requirement of a triple $(n_i, \lambda(p, i), \theta(p, i))$ is independent of $S_{\mathcal{M}_\kappa}$ if $p(i)$ is finished. We assume that $i < \|p\|$ (the case $i = \|p\|$ requires only minor modifications). Note that under this assumption we either have $v \notin \kappa(p, i+1)$ or $\kappa(p, i+1)$ has not been computed yet (i.e. $\theta(p, i+1) = \emptyset$). We distinguish these two cases:

(a) $\kappa(p, i+1)$ has not been computed yet. Then $p(i)$ is 1-level by assumption. Furthermore, $p(i)$ must be active. Let $\theta(p, i) = (\kappa(p, i), c)$ where c is a configuration of $\mathcal{M}_\kappa$. Assume first that c is not a terminal configuration, i.e. $\mathcal{M}_\kappa$ requires more computation steps. Then $\mathcal{M}$ simulates another com-

[4] Note that $\mathcal{M}_\lambda$ may need $\mathbf{\Omega}(2^{|V|})$ steps to produce a label. Depending on ϕ, this can be improved (see results on Streett games, Thm. 3). We cannot spread out this computation over several iterations (c.f. Rem. 1 on page 441).

putation step of $\mathcal{M}_\kappa$ and replaces c by its unique successor configuration c'. $\mathcal{M}$ outputs an arbitrary vertex neighboring v in $\kappa(p,i)$. This requires a number of steps in $\mathsf{poly}(|V|,|E|)$.

If c is a terminal configuration we replace c by the set $\mathcal{D}^{p(i)}$ and the integer $k_{p(i)}$, which have been computed. We then compute $\kappa(p,i+1)$ from $\mathcal{D}^{p(i)}$. If $i+1 < \|p\|$, then we also compute $\kappa(p,i+2)$. This is an attractor on the set $\lambda(p,i+1) \setminus \lambda(p,i+2)$, as $p(i+1)$ is 0-level. Finally, if even $i+2 < \|p\|$, we set $p(i+2)$ active, i.e. we replace $\theta(p,i+2)=\emptyset$ by $(\kappa(p,i+2),c_I)$. All these steps need time in $\mathsf{poly}(|V|,|E|)$.

For the next move, we distinguish $v \notin \kappa(p,i+1)$ and $v \in \kappa(p,i+1)$. If $v \notin \kappa(p,i+1)$, $\mathcal{M}$ computes the minimal t with $v \in U_t^{p(i)}$ (note that this can be done while computing $\kappa(p,i+1)$ above). Then v is either in the 0-attractor to $U_{t-1}^{p(i)}$ or $v \in Z_t^{p(i)}$. In the first case, $\mathcal{M}$ outputs according to the corresponding attractor strategy. Otherwise, $\mathcal{M}$ updates its memory state to a path through the corresponding child of $p(i)$. This requires $\boldsymbol{\mathcal{O}}(|V| \cdot T_{\mathcal{M}_\lambda})$ steps, where $T_{\mathcal{M}_\lambda}$ is the runtime of $\mathcal{M}_\lambda$. If $v \in \kappa(p,i+1)$, $\mathcal{M}$ outputs an arbitrary neighbor of v in $\kappa(p,i+1)$.

(b) $\kappa(p,i+1)$ has been computed. We proceed as the usual Zielonka strategy indicates: If $p(i)$ is 0-level, we play an attractor strategy on the set $\lambda(p,i) \setminus \lambda(p,i+1)$, updating the memory path if necessary. Otherwise, $p(i)$ is 1-level. Then we compute the minimal t with $v \in U_t^{p(i)}$ and play an attractor strategy on $U_t^{p(i)}$, or we update the memory to the leftmost path through the unique child c of $p(i)$ with $Z_t \subseteq \kappa(c)$ and output an arbitrary neighbor of v in $\kappa(c)$. Again, all of this is feasible in time in $\mathsf{poly}(|V|,|E|,T_{\mathcal{M}_\lambda})$.

The proofs of the following two lemmas can be found in [12]:

Lemma 2. *$\mathcal{M}$, as described above, implements a winning strategy for player 0.*

Note that the size of $\mathcal{M}$ depends only on the representations of $\mathcal{A}$ and ϕ it uses, as well as on $\|\mathcal{M}_\lambda\|$ and $\|\mathcal{M}_\kappa\|$. Those representations can be implemented with a polynomial number of control states. All other parts of the machine are independent of the size of $\mathcal{A}$ or ϕ. Let $T_{\mathcal{M}_\lambda}$ and $S_{\mathcal{M}_\kappa}$ denote the runtime of $\mathcal{M}_\lambda$ and the space requirement of $\mathcal{M}_\kappa$.

Lemma 3. *Let $\mathbf{G} = (\mathcal{A},\phi)$ be a Muller game, where $\mathcal{A} = (V,E)$ and ϕ is a propositional formula. There exists a strategy machine $\mathcal{M}$ of size $\|\mathcal{M}\| \in \mathsf{poly}(|V|,\|\phi\|)$, space consumption $S(\mathcal{M}) \in \mathsf{poly}(|V|,S_{\mathcal{M}_\kappa})$, and latency $T(\mathcal{M}) \in \mathsf{poly}(|V|,|E|,T_{\mathcal{M}_\lambda})$ which implements a winning strategy for player 0.*

Using that solving Muller games is in PSPACE [3,4,17] we can show $S_{\mathcal{M}_\kappa} \in \mathsf{poly}(|V|,\|\phi\|)$ (see [12]). Also, $T_{\mathcal{M}_\lambda} \in \boldsymbol{\mathcal{O}}\big(|V| \cdot \log_2(|V|) \cdot \max\{|\mathfrak{F}|,|\mathfrak{F}^{\mathscr{C}}|\}\big)$, where $\mathfrak{F}$ is the explicit condition for ϕ. For a proof of the next theorem, see [12]:

Theorem 2. *For any Muller game $\mathbf{G} = (\mathcal{A},\mathfrak{F})$, where $\mathcal{A} = (V,E)$ and $\mathfrak{F}$ is given by a propositional formula ϕ: There exists a strategy machine $\mathcal{M}$ of size $\|\mathcal{M}\| \in \mathsf{poly}(|V|,\|\phi\|)$, space consumption $S(\mathcal{M}) \in \mathsf{poly}(|V|,\|\phi\|)$, and latency $T(\mathcal{M})$ polynomial in $|V|+|E|$ and linear in* $\max\{|\mathfrak{F}|,|\mathfrak{F}^{\mathscr{C}}|\}$.

Remark 1. Unlike the situation of computing κ, we cannot spread out the computation of $\mathcal{M}_\lambda$ across the infinite play, because we do not have a compact representation of the sets $\lambda(x_0), \ldots, \lambda(x_k)$ (if $x_0, \ldots, x_k$ are siblings). For $\kappa(x_i)$ the set $\mathcal{D}^x$ provides such a compact representation.

Muller games are usually solved via *latest appearance records* [18,17], yielding a Mealy machine of size $|V| \cdot |V|!$. The straightforward simulation (c.f. Prop. 1) of this machine is of size $|V|^2 \cdot |V|!$. The size we obtain is exponentially lower, at the price of an exponentially longer latency. For Streett games we can improve the bound on $T_{\mathcal{M}_\lambda}$. Every Streett condition Ω is equivalent to $\mathfrak{F}_\Omega = \{X \subseteq V \mid X$ violates no pair in $\Omega\}$. We can show (for details, see [12]):

Proposition 3. *Let Ω be a Streett condition. Then there exists a machine $\mathcal{M}_\lambda$ computing λ with runtime polynomial in $|V|$ and $|\Omega|$.*

With Prop. 3 and Lem. 3 one shows (see [12]):

Theorem 3. *Let $\mathbf{G} = (\mathcal{A}, \Omega)$ be a Streett game. Then there exists a strategy machine $\mathcal{M}$ of size $\|\mathcal{M}\| \in \mathsf{poly}(|V|, |\Omega|)$, space consumption $S(\mathcal{M}) \in \mathsf{poly}(|V|, |\Omega|)$ and latency $T(\mathcal{M}) \in \mathsf{poly}(|V|, |\Omega|)$.*

Streett games are usually solved using *index appearance records* [19,13]. This yields a Mealy machine with $|\Omega|^2 \cdot |\Omega|!$ states. The straightforward simulation (Prop. 1) gives a strategy machine of size $\mathcal{O}(|\Omega|^2 \cdot |\Omega|! \cdot |V|)$ and latency $\mathcal{O}(|\Omega|)$. We significantly reduce the size while the latency remains polynomial in $|\Omega|$.

6 Conclusion and Future Work

We introduced the formal model of a strategy machine, based on Turing machines. We showed how different new criteria of a strategy – latency, space requirement and size – fit into this model, providing general lower bounds for the classes of Muller, Streett, and LTL games. Using this model one can obtain polynomial sized machines with polynomial space requirement implementing winning strategies in Muller games. The runtime is linear in the size of the winning condition. This machine adapts the strategy as the play proceeds and critically relies on the fact that costly computations may be spread out over the course of several iterations. We were able to show that in the special case of a Streett game the very same algorithm can be made to work with a polynomial latency in the size of the arena and of the number of Streett pairs. The space requirement and size of all our strategy machines are polynomial in the size of the winning condition (given by a propositional formula) and of the arena. Altogether, this is an exponential improvement over the straightforward way of transforming a Mealy machine into a strategy machine via a case distinction over all inputs. This approach results in a machine of exponential size in general.

We plan to extend these results to other kinds of ω-regular games. We have partial results on Request-Response games. Considering latency, space requirement, and size, we would like to find relations between the parameters indicating the nature of a trade-off. Also, infinite-state strategies lend themselves to a closer study based on our model. Finally, using an adaptive strategy raises the question of how long the adaption takes. We try to address this in current research.

Acknowledgments. The author would like thank Wolfgang Thomas and Christof Löding for their helpful comments and suggestions.

References

1. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Specify, compile, run: Hardware from psl. ENTCS 190, 3–16 (2007)
2. Dziembowski, S., Jurdzinski, M., Walukiewicz, I.: How much memory is needed to win infinite games? In: LICS 1997, IEEE Computer Society. Washington, DC (1997)
3. Hunter, P., Dawar, A.: Complexity Bounds for Regular Games (Extended Abstract). In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 495–506. Springer, Heidelberg (2005)
4. Hunter, P., Dawar, A.: Complexity bounds for muller games. Theoretical Computer Science (TCS) (2008) (submitted)
5. Horn, F.: Explicit muller games are ptime. In: FSTTCS, pp. 235–243 (2008)
6. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs. SIAM J. Comput. 29, 132–158 (1999)
7. Holtmann, M., Löding, C.: Memory Reduction for Strategies in Infinite Games. In: Holub, J., Žďárek, J. (eds.) CIAA 2007. LNCS, vol. 4783, pp. 253–264. Springer, Heidelberg (2007)
8. Gelderie, M., Holtmann, M.: Memory reduction via delayed simulation. In: iWIGP, pp. 46–60 (2011)
9. Grohe, M., Hernich, A., Schweikardt, N.: Lower bounds for processing data with few random accesses to external memory. J. ACM 56, 12:1–12:58 (2009)
10. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. 200, 135–183 (1998)
11. Madhusudan, P.: Synthesizing reactive programs. In: Proceedings of Comp. Sci. Log., CSL 2011, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 428–442 (2011)
12. Gelderie, M.: Strategy machines and their complexity. Technical Report AIB-2012-04, RWTH Aachen University (2012), http://sunsite.informatik.rwth-aachen.de/Publications/AIB/2012/2012-04.pdf
13. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata logics, and infinite games: a guide to current research. Springer, New York (2002)
14. Löding, C.: Infinite games and automata theory. In: Apt, K.R., Grädel, E. (eds.) Lectures in Game Theory for Computer Scientists, Cambridge UP (2011)
15. Perrin, D., Pin, J.: Infinite words: automata, semigroups, logic and games. In: Pure and Applied Mathematics. Elsevier (2004)
16. Büchi, J.R., Landweber, L.H.: Solving Sequential Conditions by Finite-State Strategies. Trans. of the AMS 138, 295–311 (1969)
17. McNaughton, R.: Infinite games played on finite graphs. Annals of Pure and Applied Logic 65(2), 149–184 (1993)
18. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, pp. 60–65 (1982)
19. Safra, S.: Exponential determinization for ω-automata with strong-fairness acceptance condition (extended abstract). In: STOC 1992, pp. 275–282 (1992)

Coloring Graphs Characterized by a Forbidden Subgraph*

Petr A. Golovach[1], Daniël Paulusma[2], and Bernard Ries[3]

[1] Department of Informatics, University of Bergen, Norway
petr.golovach@ii.uib.no

[2] School of Engineering and Computing Sciences, Durham University
daniel.paulusma@durham.ac.uk

[3] Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Decision,
Université Paris Dauphine
bernard.ries@dauphine.fr

Abstract. The COLORING problem is to test whether a given graph can be colored with at most k colors for some given k, such that no two adjacent vertices receive the same color. The complexity of this problem on graphs that do not contain some graph H as an induced subgraph is known for each fixed graph H. A natural variant is to forbid a graph H only as a subgraph. We call such graphs strongly H-free and initiate a complexity classification of COLORING for strongly H-free graphs. We show that COLORING is NP-complete for strongly H-free graphs, even for $k = 3$, when H contains a cycle, has maximum degree at least five, or contains a connected component with two vertices of degree four. We also give three conditions on a forest H of maximum degree at most four and with at most one vertex of degree four in each of its connected components, such that COLORING is NP-complete for strongly H-free graphs even for $k = 3$. Finally, we classify the computational complexity of COLORING on strongly H-free graphs for all fixed graphs H up to seven vertices. In particular, we show that COLORING is polynomial-time solvable when H is a forest that has at most seven vertices and maximum degree at most four.

1 Introduction

Graph coloring involves the labeling of the vertices of some given graph by integers called colors such that no two adjacent vertices receive the same color. The corresponding COLORING problem is to decide whether a graph can be colored with at most k colors for some given integer k. Due to the fact that COLORING is NP-complete for any fixed $k \geq 3$, there has been considerable interest in studying its complexity when restricted to certain graph classes. General motivation, background and related work on coloring problems restricted to special graph classes can be found in several surveys [13,14].

We study the complexity of the COLORING problem restricted to graph classes defined by forbidding a graph H as a (not necessarily induced) subgraph. So far,

* The work was supported by EPSRC (EP/G043434/1) and ERC (267959).

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 443–454, 2012.

Coloring has not been studied in the literature as regards to such graph classes. Before we summarize some related results and present our results, we first state the necessary terminology and notations.

Terminology. We consider finite undirected graphs without loops and multiple edges. We refer to the textbook of Diestel [5] for any undefined graph terminology. Let $G = (V, E)$ be a graph. The subgraph of G induced by a subset $U \subseteq V$ is denoted $G[U]$. The graph $G - u$ is obtained from G by removing vertex u. For a vertex u of G, its *open neighborhood* is $N(u) = \{v \mid uv \in E\}$, its *closed neighborhood* is $N[u] = N(u) \cup \{u\}$, and its degree is $d(u) = |N(u)|$. The maximum degree of G is denoted by $\Delta(G)$ and the minimum degree by $\delta(G)$. The *distance* $\mathrm{dist}(u, v)$ between two vertices u and v of G is the number of edges of a shortest path between them. The *girth* $g(G)$ is the length of a shortest cycle in G. We say that G is *(strongly) H-free* for some graph H if G has no subgraph isomorphic to H; note that this is more restrictive than forbidding H as an induced subgraph. A *subdivision* of an edge $uv \in E$ is the operation that removes uv and adds a new vertex adjacent to u and v. A graph H is a *subdivision* of G if H is obtained from G by a sequence of edge subdivisions. A *coloring* of G is a mapping $c : V \to \{1, 2, \ldots\}$, such that $c(u) \neq c(v)$ if $uv \in E$. We call $c(u)$ the *color* of u. A *k-coloring* of G is a coloring c of G with $1 \leq c(u) \leq k$ for all $u \in V$. If G has a k-coloring, then G is called *k-colorable.* The *chromatic number* $\chi(G)$ is the smallest integer k such that G is k-colorable. The k-Coloring problem is to test whether a graph admits a k-coloring for some fixed integer k. If k is in the input, then we call this problem Coloring. The graph P_n is the path on n vertices.

Related Work. Král', Kratochvíl, Tuza and Woeginger [11] completely determined the computational complexity of Coloring for graph classes characterized by a forbidden *induced* subgraph and achieved the following dichotomy. Here, $P_1 + P_3$ denotes the disjoint union of P_1 and P_3.

Theorem 1 ([11]). *If some fixed graph H is a (not necessarily proper) induced subgraph of P_4 or of $P_1 + P_3$, then* Coloring *is polynomial-time solvable on graphs with no induced subgraph isomorphic to H; otherwise it is* NP*-complete on this graph class.*

The complexity classification of the k-Coloring problem for graphs with no induced subgraphs isomorphic to some fixed graph H is still open. For $k = 3$, it has been classified for graphs H up to six vertices [3], and for $k = 4$ for graphs H up to five vertices [7]. We refer to the latter paper for a survey on the complexity status of k-Coloring for graph classes characterized by a forbidden induced subgraph.

Our Results. Recall that a strongly H-free graph denotes a graph with no subgraph isomorphic to some fixed graph H. Forbidding a graph H as an induced subgraph is equivalent to forbidding H as a subgraph if and only if H is a complete graph (a graph with an edge between any two distinct vertices). Hence, Theorem 1 tells us that Coloring is NP-complete for strongly H-free graphs if H is a complete graph. We extend this result by proving the following two theorems in Sections 2 and 3, respectively; note that the case when H is a complete graph is covered by condition (a) of Theorem 2. The trees $T_1, \ldots, T_6$ are displayed in Figure 1. For an integer $p \geq 0$, the graph T_2^p is the graph obtained from T_2 after subdividing the edge st p times; note that $T_2^0 = T_2$.

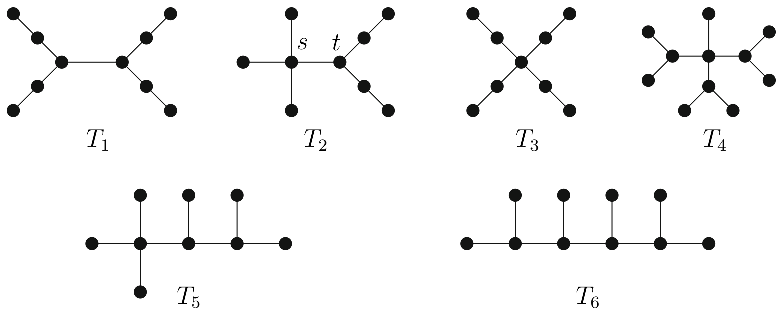


Fig. 1. The trees $T_1, \ldots, T_6$

Theorem 2. 3-Coloring *(and hence* Coloring*) is* NP*-complete for strongly H-free graphs if*

(a) H contains a cycle, or
(b) $\Delta(H) \geq 5$, or
(c) H has a connected component with at least two vertices of degree four, or
(d) H contains a subdivision of the tree T_1 as a subgraph, or
(e) H contains the tree T_2^p as a subgraph for some $0 \leq p \leq 9$, or
(f) H contains one of the trees T_3, T_4, T_5, T_6 as a subgraph.

Theorem 3. Coloring *is polynomial-time solvable for strongly H-free graphs if*

(a) H is a forest with $\Delta(H) \leq 3$, such that each connected component has at most one vertex of degree 3, or
(b) H is a forest with $\Delta(H) \leq 4$ and $|V_H| \leq 7$.

Theorems 1–3 tell us that the Coloring problem behaves differently on graphs characterized by forbidding H as an induced subgraph or as a subgraph. As a consequence of Theorems 2 and 3(b) we can classify the Coloring problem on strongly H-free graphs for graphs H up to 7 vertices. The problem is NP-complete if H is not a forest or $\Delta(H) \geq 5$, and polynomial-time solvable otherwise.

Future Work. The aim is to complete the computational complexity classification of Coloring for strongly H-free graphs. Our current proof techniques are rather diverse, and a more unifying approach may be required.

2 The Proof of Theorem 2

In the remainder of the paper we write H-free instead of strongly H-free as a shorthand notation. Below we include the proofs of a number of cases of Theorem 2. The remaining cases are left out due to space restrictions.

(a) Maffray and Preissmann [12] showed that 3-Coloring is NP-complete for triangle-free graphs. This result has been extended by Kamiński and Lozin [10], who proved that k-Coloring is NP-complete for the class of graphs of girth at least p for any fixed $k \geq 3$ and $p \geq 3$. Suppose that H contains a cycle. Then $g(H)$ is finite. Let $p = g(H) + 1$. It remains to observe that any graph of girth at least p does not contain H as a subgraph, and (a) follows.

(b) It is well known that 3-Coloring is NP-complete for graphs of maximum degree at most four [6]. Then, because any graph G with $\Delta(G) \leq 4$ does not contain a graph H with $\Delta(H) \geq 5$ as a subgraph, (b) holds.

(c) As before, we reduce from 3-Coloring for graphs of maximum degree at most four. Let $G = (V, E)$ be a graph of maximum degree at most four. We define a useful graph operation. In order to do this, we need the graph displayed in Figure 2. It has vertex set $\{x, y, z, t\}$ and edge set $\{xz, xt, yz, yt, zt\}$ and is called a *diamond* with *poles* x, y. We observe that in any 3-coloring of a diamond with poles x, y, the vertices x and y are colored alike.

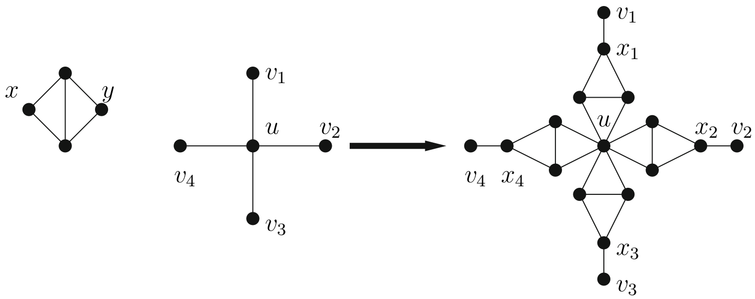


Fig. 2. A diamond with poles x, y and the vertex-diamond operation

The graph operation that we use is displayed in Figure 2. For a vertex $u \in V$ with four neighbors $v_1, \ldots, v_4$, we do as follows. We delete the edges uv_i for $i = 1, \ldots, 4$. We then add 4 diamonds with poles x_i, y_i for $i = 1, \ldots, 4$ and identify u with each y_i. Finally, we add the edges $v_i x_i$ for $i = 1, \ldots, 4$. We call this operation the *vertex-diamond* operation. Note that this operation is only defined on vertices of degree four. Because any 3-coloring gives the poles of a diamond the same color, the resulting graph is 3-colorable if and only if G is 3-colorable. We also observe that this operation when applied on a vertex u

increases the distance between u and any other vertex of G by 2. Moreover, the new vertices added have degree three.

To complete the proof of (c), let H be a graph that has a connected component D with at least two vertices of degree four. Let α denote the maximum distance between two such vertices in D. Then we apply α vertex-diamond operations on each vertex of degree four in G. By our previous observations, the resulting graph G^* is D-free, and consequently, H-free, and in addition, G^* is 3-colorable if and only if G is 3-colorable. Hence (c) holds.

Subcases $\mathbf{p = 0}$ and $\mathbf{p = 1}$ of (e) and Case $\mathbf{T_5}$ of (f). As before, we reduce from 3-Coloring for graphs of maximum degree at most four. Let $G = (V, E)$ be a graph of maximum degree at most four. We construct the graph G^* defined in case (c). We observe that G^* is T_2^0-free, T_2^1-free and T_5-free, because every vertex of degree at least four in G^* is obtained by identifying pole vertices of diamonds. Hence, the subcases $p = 0$ and $p = 1$ of (e) and the corresponding subcase of (f) hold.

3 The Proof of Theorem 3

Let G be a graph. A graph H is a *minor* of G if H can be obtained from a subgraph of G by a sequence of edge contractions. We first prove Theorem 3(a).

Theorem 3(a). *Let H be a fixed forest with $\Delta(H) \leq 3$, such that each connected component of H has at most one vertex of degree three. Then* Coloring *can be solved in polynomial time for H-free graphs.*

Proof. Let G be a graph. If $|V_H| = 1$, then the statement of the theorem holds. Suppose that $|V_H| \geq 2$. Let $H_1, \ldots, H_p$ be the connected components of H. Consider a connected component H_i for some $1 \leq i \leq p$. Because $\delta(H) \leq 3$, we find that $\delta(H_i) \leq 3$. Moreover, by our assumption, H_i contains at most one vertex of degree 3. Then H_i is either a path or a subdivided star, in which the centre vertex has degree 3. As such, H_i is a subgraph of G if and only if H_i is a minor of G. Consequently, H is a subgraph of a graph G if and only if H is a minor of G. By a result of Bienstock et al. [2], every graph that does not contain H as a minor has path-width at most $|V_H| - 2$. Hence G has path-width at most $|V_H| - 2$. Because H is fixed, this implies that G has bounded path-width, and consequently, bounded treewidth. Because Coloring can be solved in linear time for graphs of bounded treewidth as shown by Arnborg and Proskurowski [1], the result follows. □

Theorem 3(a) limits the remaining cases of Theorem 3(b) to those graphs H that are a forest on at most 7 vertices and that contain a vertex of degree 4 or two vertices of degree at least 3. Moreover, our goal is to show polynomial-time solvability for such cases, and a graph is H-free if it is H'-free for any subgraph H' of H. This narrows down our case analysis to the trees $H_1, \ldots, H_5$ shown in Fig. 3. We consider each such tree, but we first give some auxiliary results.

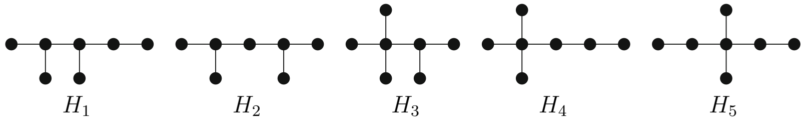


Fig. 3. The trees $H_1, \ldots, H_5$

Observation 1. *Let G be a graph with $|V_G| \geq 2$. Let $u \in V_G$ with $d_G(u) < k$ for some integer $k \geq 1$. Then G is k-colorable if and only if $G - u$ is k-colorable.*

We say that a vertex u of a graph G is *universal* if $G = G[N_G[u]]$, i.e., if u is adjacent to all other vertices of G.

Observation 2. *Let u be a universal vertex of a graph G with $|V_G| \geq 2$. Let $k \geq 2$ be an integer. Then G is k-colorable if and only if $G - u$ is $(k-1)$-colorable.*

A vertex u of a connected graph G with at least two vertices is a *cut-vertex* if $G - u$ is disconnected. A maximal connected subgraph of G with no cut-vertices is called a *block* of G.

Observation 3. *Let G be a connected graph, and let k be a positive integer. Then G is k-colorable if and only if each block of G is k-colorable.*

Let (G, k) be an instance of Coloring. We apply the following preprocessing rules recursively, and as long as possible. If after the application of a rule we can apply some other rule with a smaller index, then we will do this.

Rule 1. Find all connected components of G and consider each of them.

Rule 2. Check if G is 1-colorable or 2-colorable. If so, then stop considering G.

Rule 3. If $|V_G| \geq 2$, $k \geq 3$, and G has a vertex u with $d_G(u) \leq 2$, take $(G - u, k)$.

Rule 4. If $|V_G| \geq 2$, $k \geq 3$, and G has a universal vertex u, take $(G - u, k - 1)$.

Rule 5. If G is connected, then find all blocks of G and consider each of them.

We obtain the following lemma. Its proof has been omitted due to space restrictions.

Lemma 1. *Let G be an n-vertex graph that together with an integer $k \geq 3$ forms an instance of* Coloring. *Applying rules 1–5 recursively and exhaustively takes polynomial time and yields a set I of at most n instances, such that (G, k) is a yes-instance if and only if every instance of I is a yes-instance. Moreover, each $(G', k') \in I$ has the following properties:*

(i) $|V_{G'}| \leq |V_G|$;
(ii) $\delta(G') \geq 3$;
(iii) *G' has no universal vertices;*
(iv) *G' is 2-connected;*
(v) $3 \leq k' \leq k$;
(vi) *if G is H-free for some graph H, then G' is H-free as well.*

3.1 The Cases $H = H_1$ and $H = H_2$

We first give some extra terminology. Let $G = (V, E)$ be a graph. We let $\omega(G)$ denote the size of a maximum clique in G. The *complement* of G is the graph $\overline{G}$ with vertex set V, such that any two distinct vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G. If $\chi(F) = \omega(F)$ for any induced subgraph F of G, then G is is called *perfect*. Let C_r denote the cycle on r vertices. We will use the Strong Perfect Graph Theorem proved by Chudnovsky et al. [4]. This theorem tells us that a graph is perfect if and only if it does not contain C_r or $\overline{C}_r$ as an induced subgraph for any odd integer $r \geq 5$.

Lemma 2. *Let G be a 2-connected graph with $\delta(G) \geq 3$ that has no universal vertices. If G is H_1-free or H_2-free, then G is perfect.*

Proof. Note that H_1 and H_2 are both subgraphs of $\overline{C}_r$ for any $r \geq 7$. Moreover, $C_5 = \overline{C_5}$. Then, by the Strong Perfect Graph Theorem [4], we are left to prove that G contains no induced cycle C_r for any odd integer $r \geq 5$. To obtain a contradiction, assume that G does contain an induced cycle $C = v_0 v_1 \cdots v_{r-1} v_{r-1} v_0$ for some odd integer $r \geq 5$.

First suppose that G is H_1-free. Let $0 \leq i \leq r - 1$ and consider the path $v_i v_{i+1} \cdots v_{i+3} v_{i+4}$, where the indices are taken modulo r. Since $\delta(G) \geq 3$, v_{i+1} and v_{i+2} each have at least one neighbor in $V' = V \setminus \{v_0, \ldots, v_{r-1}\}$, say v_{i+1} is adjacent to some vertex u and v_{i+2} is adjacent to some vertex v. Because G is H_1-free, $u = v$, and moreover, $|N(v_{i+1}) \cap V'| = |N(v_{i+2}) \cap V'| = 1$. Because $0 \leq i \leq r - 1$ was taken arbitrarily, we deduce that the vertices $v_0, \ldots, v_{r-1}$ are all adjacent to the same vertex $u \in V'$ and to no other vertices in V'. Because G is 2-connected, u is not a cut-vertex. Hence, $V' = \{u\}$. However, then u is a universal vertex. This is a contradiction.

Now suppose that G is H_2-free. By the same arguments and the fact that r is odd, we conclude again that there exists a universal vertex $u \in V'$. This is a contradiction. □

We are now ready to prove that Coloring is polynomial-time solvable for H_1-free and for H_2-free graphs. Let G be a graph, and let $k \geq 1$ be an integer. If $k \leq 2$, then Coloring is even polynomial-time solvable for general graphs. Suppose that $k \geq 3$. Then, by Lemma 1, we may assume without loss of generality that G is 2-connected, has $\delta(G) \geq 3$ and does not contain any universal vertices. Lemma 2 then tells us that G is perfect. Because Grötschel et al. [8] showed that Coloring is polynomial-time solvable for perfect graphs, our result follows.

3.2 The Case $H = H_3$

We first give some additional terminology. We say that we *identify* two distinct vertices $u, v \in V_G$ if we first remove u, v and then add a new vertex w by making it (only) adjacent to the vertices of $(N_G(u) \cup N_G(v)) \setminus \{u, v\}$.

Consider the graphs $F_1, \ldots, F_4$ shown in Fig. 4. We call the vertices x_1, x_2 of F_1, x_1, x_2, x_3 of F_2 and x_1, x_2, y_1, y_2 of F_3 and F_4 the *pole vertices* of the corresponding graph F_i, whereas the other vertices of F_i are called *centre vertices*. We say that a graph G *properly contains* F_i for some $1 \leq i \leq 4$ if G contains F_i as an induced subgraph, in such a way that centre vertices of F_i are only adjacent to vertices of F_i, i.e., the subgraph F_i is connected to other vertices of G only via its poles.

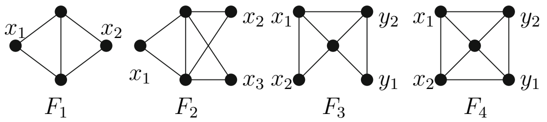


Fig. 4. The graphs F_1, F_2, F_3, F_4

For our result we need one additional rule that we apply on a graph G.

Rule 6. If G properly contains F_i for some $1 \leq i \leq 4$, then remove the centre vertices of F_i from G and identify the pole vertices of F_i as follows:

- if $i = 1$, then identify x_1 and x_2;
- if $i = 2$, then identify x_1, x_2, and x_3;
- if $i = 3$ or $i = 4$, then identify x_1 and y_1, and also identify x_2 and y_2.

We prove that Coloring can be solved in polynomial time for H_3-free graphs as follows. Let G be an H_3-free graph, and let $k \geq 1$ be an integer.

Case 1. $k \leq 2$.
Then Coloring can be solved in polynomial time even for general graphs.

Case 2. $k \geq 3$.
By Lemma 1, we may assume without loss of generality that $\delta(G) \geq 3$ and that G contains no universal vertices. In Lemma 3 we show that $\Delta(G) \leq 4$. Then Brooks' Theorem (Theorem 5.2.4 in [5]) tells us that G is 4-colorable unless $G = K_5$. In the latter case, G is 5-colorable.

Case 2a. $k \geq 5$.
Then (G, k) is a yes-answer.

Case 2b. $k = 4$.
Then (G, k) is a yes-answer if and only if $G \neq K_5$.

Case 2c. $k = 3$.
We show in Lemma 4 (stated on the next page) that an application of Rule 6 on G yields an H_3-free graph that is 3-colorable if and only if G is 3-colorable. We apply Rule 6 exhaustively. This takes polynomial time, because each application of Rule 6 takes linear time and reduces the size of G. In order to maintain the properties of having minimum degree at least 3 and containing no universal vertices, we apply Lemma 1 after each application of Rule 6. Hence, afterward, we have found in polynomial time a (possibly empty) set $\mathcal{G}$ of at most n graphs,

such that G is 3-colorable if and only if each graph in $\mathcal{G}$ is 3-colorable. Moreover, each $G' \in \mathcal{G}$ is H_3-free, has minimum degree at least 3, contains no universal vertices, and in addition, does not properly contain any of the graphs $F_1, \ldots, F_4$. Then, by Lemma 3, each $G' \in \mathcal{G}$ has $\Delta(G') \leq 4$. As a consequence, we may apply Lemma 5. This lemma tells us that a graph $G' \in \mathcal{G}$ is 3-colorable if and only if it does not contain K_4 as a subgraph. As we can check the latter condition in polynomial time and $|\mathcal{G}| \leq n$, i.e., we have at most n graphs to check, our result follows.

What is left to do is to state and prove Lemmas 3–5. We omitted the proof of Lemma 4.

Lemma 3. *Let G be an H_3-free graph with no universal vertices. If $\delta(G) \geq 3$, then $\Delta(G) \leq 4$.*

Proof. Let $G = (V, E)$ be an H_3-free graph with no universal vertices. Suppose that $\delta(G) \geq 3$. To obtain a contradiction assume that $d_G(u) \geq 5$ for some vertex $u \in V$. Because G has no universal vertices, there is a vertex $v \in N_G(u)$ such that v has a neighbor $x \in V \setminus N_G[u]$. Because $d_G(v) \geq \delta(G) \geq 3$, we deduce that v has another neighbor $y \notin \{u, x\}$. Because $d_G(u) \geq 5$, we also deduce that u has three neighbors z_1, z_2, z_3 neither equal to v nor to y. However, the subgraph of G with vertices $u, v, x, y, z_1, z_2, z_3$ and edges $uz_1, uz_2, uz_3, uv, vx, vy$ is isomorphic to H_3. This is a contradiction, because G is H_3-free. □

Lemma 4. *Let G be an H_3-free graph with $\delta(G) \geq 3$ and $\Delta(G) \leq 4$. Let G' be the graph obtained from G after one application of Rule 6. Then G' is 3-colorable if and only if G is 3-colorable. Moreover, G' is H_3-free.*

Lemma 5. *Let G be an H_3-free graph with $\delta(G) \geq 3$ and $\Delta(G) \leq 4$ that does not properly contain any of the graphs $F_1, \ldots, F_4$. Then G is 3-colorable if and only if G is K_4-free.*

Proof. Let $G = (V, E)$ be an H_3-free graph with $\delta(G) \geq 3$ and $\Delta(G) \leq 4$ that does not properly contain any of the graphs $F_1, \ldots, F_4$. First suppose that G is 3-colorable. This immediately implies that G is K_4-free.

Now suppose that G is K_4-free. If $\Delta(G) \leq 3$, then Brooks' Theorem (cf. [5]) tells us that G is 3-colorable unless $G = K_4$, which is not the case. Hence, we may assume that G contains at least one vertex of degree four. To obtain a contradiction, assume that G is a minimal counter-example, i.e., $\chi(G) \geq 4$ and $G - v$ is 3-colorable for all $v \in V$.

Let u be a vertex of degree four in G, and let $N_G(u) = \{v_1, v_2, v_3, v_4\}$. We first show the following four claims.

(a) $G[N_G(u)]$ is C_3-free;
(b) $G[N_G(u)]$ contains no vertex of degree three;
(c) $G[N_G(u)]$ is not isomorphic to P_4;
(d) $G[N_G(u)]$ is not isomorphic to C_4.

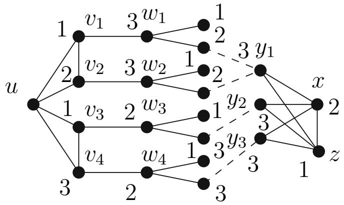


Fig. 5. The structure of the graph G. We note that neighbors of $w_1, \ldots, w_4$ not equal to $v_1, \ldots, v_4$ may not be distinct.

Claims (a)–(d) can be seen as follows. If $G[N_G(u)]$ contains C_3 as a subgraph, then $G[N_G[u]]$, and consequently, G contains K_4 is a subgraph of G. This proves (a). If $G[N_G(u)]$ contains a vertex of degree three, then G properly contains F_2 as G is C_3-free due to (a). This proves (b). If $G[N_G(u)]$ is isomorphic to P_4, then G properly contains F_3. This proves (c). If $G[N_G(u)]$ is isomorphic to C_4, then G properly contains F_4. This proves (d).

Because G is H_3-free, each v_j has at most one neighbor in $V \setminus N_G[u]$. Because $\delta(G) \geq 3$, this means that $G[N_G(u)]$ contains no isolated vertices. Then, by claims (a)–(d), we find that $G[N_G(u)]$ contains exactly two edges. Moreover, $d_G(v_j) = 3$ for $j \in \{1, \ldots, 4\}$ as $\delta(G) \geq 3$.

We assume without loss of generality that v_1v_2 and v_3v_4 are edges in G. Let w_j be the neighbor of v_j in $V \setminus N_G[u]$ for $j = 1, \ldots, 4$. We note that $w_1 \neq w_2$ and $w_3 \neq w_4$, as otherwise G properly contains F_1.

Recall that $G - u$ is 3-colorable. Let c be an arbitrary 3-coloring of $G - u$. We show that the following two claims are valid for c up to a permutation of the colors 1, 2, 3.

(1) $c(v_1) = c(v_3) = 1$, $c(v_2) = 2$ and $c(v_4) = 3$;
(2) $c(w_1) = c(w_2) = 3$ and $c(w_3) = c(w_4) = 2$;

Claims (1) an (2) can be seen as follows. If c uses at most two different colors on $v_1, \ldots, v_4$, then we can extend c to a 3-coloring of G, which is not possible as $\chi(G) \geq 4$. Hence, c uses three different colors on $v_1, \ldots, v_4$. Then we may assume without loss of generality that $c(v_1) = c(v_3) = 1$, $c(v_2) = 2$ and $c(v_4) = 3$. This proves (1). We now prove (2). In order to obtain a contradiction, assume that $c(w_1) \neq c(w_2)$. Because $c(v_2) = 2$, we find that $c(w_2) = 1$ or $c(w_2) = 3$. If $c(w_2) = 1$, then we change the color of v_2 into 3, contradicting (1). Hence, $c(w_2) = 3$. Then, as $c(v_1) = 1$, we obtain $c(w_1) = 2$. However, we can now change the colors of v_1 and v_2 into 3 and 1, respectively, again contradicting (1). We conclude that $c(w_1) = c(w_2)$. Hence, $c(w_1) = c(w_2) = 3$. By the same arguments, we find that $c(w_3) = c(w_4)$. Hence, $c(w_3) = c(w_4) = 2$. This proves (2).

The facts that $w_1 \neq w_2$ and $w_3 \neq w_4$ together with Claim (2) imply that w_1, w_2, w_3, w_4 are four distinct vertices. We observe that $d_G(w_j) = 3$ for $j = 1, \ldots, 4$, as otherwise H_3 is a subgraph of G. See Fig. 5 for an illustration. In this figure we also indicate that w_1, w_2 have neighbors colored with colors 1 and

2, and that w_3, w_4 have neighbors colored with colors 1 and 3, as otherwise we could recolor $w_1, \ldots, w_4$ such that $c(w_1) \neq c(w_2)$ or $c(w_3) \neq c(w_4)$, and hence we would contradict Claim (2). We may also assume without loss of generality that c is chosen in such a way that the set of vertices with color 1 is maximal, i.e., each vertex with color 2 or 3 has a neighbor with color 1.

Consider the subgraph Q of $G-u$ induced by the vertices colored with colors 2 and 3. We claim that the vertices w_1 and v_2 are in the same connected component of Q. To show this, suppose that there is a connected component Q' of Q that contains w_1 but not v_2. Then we recolor all vertices of Q' colored 2 with color 3 and all vertices of Q' colored 3 with color 2. We obtain a 3-coloring of $G-u$ such that w_1 and w_2 are colored by distinct colors, contradicting Claim (2). Using the same arguments, we conclude that w_3 and v_4 are in the same connected component of Q. Now we show that all the vertices w_1, v_2, w_3, v_4 are in the same connected component of Q. Suppose that there is a connected component Q' of Q that contains w_1, v_2 but not w_3, v_4. Then we recolor all vertices of Q' colored 2 with color 3 and all vertices colored 3 with color 2. We obtain a 3-coloring of $G-u$ such that w_1, w_2, w_3, w_4 are colored with the same color, contradicting Claim (2).

We observe that $d_Q(w_1) = d_Q(v_2) = d_Q(w_3) = d_Q(v_4) = 1$. Then, because w_1, v_2, w_3, v_4 belong to the same connected component of Q, we find that Q contains a vertex x with $d_Q(x) \geq 3$.

Let $y_1, \ldots, y_r$ denote the neighbors of x in Q for some $r \geq 3$. Because $y_1, \ldots y_r$ are colored with the same color, they are pairwise non-adjacent. Because $\Delta(G) \leq 4$, we find that $r \leq 4$. First suppose that $r = 4$. Because $d_G(y_1) \geq 3$ as $\delta(G) \geq 3$ and $y_1, \ldots, y_4$ are pairwise non-adjacent, y_1 has at least two neighbors in $V \setminus N_G[x]$. However, then G contains H_3 as a subgraph. This is a contradiction. Now suppose that $r = 3$. Recall that the set of vertices with color 1 is maximal. Hence x is adjacent to a vertex z with color 1. Because G is H_3-free and $d_G(y_i) \geq 3$ for $i = 1, 2, 3$, we find that z is adjacent to y_1, y_2, y_3. However, since $\Delta(G) \leq 4$, this means that $G[N_G[z]]$ is isomorphic to F_2. Consequently, G properly contains F_2. This contradiction completes the proof of Lemma 5. □

3.3 The Cases $H = H_4$ and $H = H_5$

For these cases we replace Rule 4 by a new rule. Let $G = (V, E)$ be a graph and k be an integer.

Rule 4*. If $k \geq 3$ and $V \setminus N_G[u]$ is an independent set for some $u \in V$, take $(G[N_G(u)], k-1)$.

We prove that Coloring can be solved in polynomial time for H_4-free graphs and for H_5-free graphs in the following way. Let $G = (V, E)$ be a graph, and let $k \geq 1$ be an integer. If $k \leq 2$, then Coloring can be solved in polynomial time even for general graphs. Now suppose that $k \geq 3$. Lemma 6 (stated on the next page) shows that Rule 4* is correct. Moreover, an application of Rule 4* takes linear time and reduces the number of vertices of G by at least one. Hence, we can replace Rule 4 by Rule 4* in Lemma 1. Due to this, we may assume without loss of generality that G is 2-connected and has $\delta(G) \geq 3$, and moreover, that $V \setminus N_G[u]$ contains at least two adjacent vertices for all $u \in V$. Then Lemma 7 tells us that $\Delta(G) \leq 3$. By using Brooks' Theorem (cf. [5]) we find that G is 3-colorable, unless $G = K_4$. Hence, (G, k) is a yes-answer when $k \geq 4$, whereas (G, k) is a yes-answer when $k = 3$ if and only if $G \neq K_4$.

What is left to do is to state and prove Lemmas 6 and 7. We omitted both proofs.

Lemma 6. *Let $k \geq 2$ be an integer, and let u be a vertex of a graph $G = (V, E)$ such that $V \setminus N_G[u]$ is an independent set. Then G is k-colorable if and only if $G[N_G(u)]$ is $(k-1)$-colorable.*

Lemma 7. *Let $G = (V, E)$ be a 2-connected graph with $\delta(G) \geq 3$ such that $V \setminus N_G[u]$ contains at least two adjacent vertices for all $u \in V$. If G is H_4-free or H_5-free, then $\Delta(G) \leq 3$.*

References

1. Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees. Discrete Applied Mathematics 23, 11–24 (1989)
2. Bienstock, D., Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a forest. J. Comb. Theory, Ser. B 52, 274–283 (1991)
3. Broersma, H.J., Golovach, P.A., Paulusma, D., Song, J.: Updating the complexity status of coloring graphs without a fixed induced linear forest. Theoretical Computer Science 414, 9–19 (2012)
4. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The strong perfect graph theorem. Annals of Mathematics 164, 51–229 (2006)
5. Diestel, R.: Graph Theory, Electronic Edition. Springer (2005)
6. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete problems. In: Proceedings of the sixth annual ACM Symposium on Theory of Computing (STOC 1974), pp. 47–63 (1974)
7. Golovach, P.A., Paulusma, D., Song, J.: 4-Coloring H-Free Graphs When H Is Small. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 289–300. Springer, Heidelberg (2012)
8. Grötschel, M., Lovász, L., Schrijver, A.: Polynomial algorithms for perfect graphs. Ann. Discrete Math., Topics on Perfect Graphs 21, 325–356 (1984)
9. Kamiński, M., Lozin, V.V.: Vertex 3-colorability of claw-free graphs. Algorithmic Operations Research 2, 15–21 (2007)
10. Kamiński, M., Lozin, V.V.: Coloring edges and vertices of graphs without short or long cycles. Contributions to Discrete Math. 2, 61–66 (2007)
11. Král, D., Kratochvíl, J., Tuza, Z., Woeginger, G.J.: Complexity of Coloring Graphs without Forbidden Induced Subgraphs. In: Brandstädt, A., Van Le, B. (eds.) WG 2001. LNCS, vol. 2204, pp. 254–262. Springer, Heidelberg (2001)
12. Maffray, F., Preissmann, M.: On the NP-completeness of the k-colorability problem for triangle-free graphs. Discrete Math. 162, 313–317 (1996)
13. Randerath, B., Schiermeyer, I.: Vertex colouring and forbidden subgraphs - a survey. Graphs Combin. 20, 1–40 (2004)
14. Tuza, Z.: Graph colorings with local restrictions - a survey. Discuss. Math. Graph Theory 17, 161–228 (1997)

Obtaining Planarity by Contracting Few Edges*

Petr A. Golovach[1], Pim van 't Hof[1], and Daniël Paulusma[2]

[1] Department of Informatics, University of Bergen, Norway
{petr.golovach,pim.vanthof}@ii.uib.no
[2] School of Engineering and Computing Sciences, Durham University, UK
daniel.paulusma@durham.ac.uk

Abstract. The Planar Contraction problem is to test whether a given graph can be made planar by using at most k edge contractions. This problem is known to be NP-complete. We show that it is fixed-parameter tractable when parameterized by k.

1 Introduction

Numerous problems in algorithmic graph theory, with a large variety of applications in different fields, can be formulated as graph modification problems. A graph modification problem takes as input an n-vertex graph G and an integer k, and the question is whether G can be modified into a graph that belongs to a prescribed graph class, using at most k operations of a certain specified type. Some of the most common graph operations that are used in this setting are vertex deletions, edge deletions and edge additions, leading to famous problems such as Feedback Vertex Set, Odd Cycle Transversal, Minimum Fill-In and Cluster Editing, to name but a few. More recently, the study of graph modification problems allowing only *edge contractions* has been initiated, yielding several results that we will survey below. The contraction of an edge removes both end-vertices of an edge and replaces them by a new vertex, which is made adjacent to precisely those vertices that were adjacent to at least one of the two end-vertices. Choosing edge contraction as the only permitted operation leads to the following decision problem, for each graph class $\mathcal{H}$.

$\mathcal{H}$-Contraction
Instance: A graph G and an integer k.
Question: Does there exist a graph $H \in \mathcal{H}$ such that G can be contracted to H, using at most k edge contractions?

Heggernes et al. [8] presented a $2^{k+o(k)} + n^{O(1)}$ time algorithm for $\mathcal{H}$-Contraction when $\mathcal{H}$ is the class of paths. Moreover, they showed that in this case the problem has a linear vertex kernel. When $\mathcal{H}$ is the class of trees, they showed that the problem can be solved in $4.98^k n^{O(1)}$ time, and that a polynomial kernel does not exist unless NP $\subseteq$ coNP/poly. When the input graph is a chordal graph

* This paper is supported by EPSRC (EP/G043434/1), Royal Society (JP100692), the Research Council of Norway (197548/F20), and ERC (267959).

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 455–466, 2012.

with n vertices and m edges, then $\mathcal{H}$-Contraction can be solved in $O(n+m)$ time when $\mathcal{H}$ is the class of trees and in $O(nm)$ time when $\mathcal{H}$ is the class of paths [7]. Heggernes et al. [9] proved that $\mathcal{H}$-Contraction is fixed-parameter tractable when $\mathcal{H}$ is the class of bipartite graphs and k is the parameter. This also follows from a more general result from a recent paper by Marx, O'Sullivan, and Razgon [14] on generalized bipartization. Golovach et al. [5] considered the class of graphs of minimum degree at least d for some integer d. They showed that in this case $\mathcal{H}$-Contraction is fixed-parameter tractable when both d and k are parameters, W[1]-hard when k is the parameter, and para-NP-complete when d is the parameter.

The combination of planar graphs and edge contractions has been studied before in a closely related setting. Kamiński, Paulusma and Thilikos [10] showed that for every fixed graph H, there exists a polynomial-time algorithm for deciding whether a given planar graph can be contracted to H. Very recently, this result was improved by Kamiński and Thilikos [11]. They showed that, given a graph H and a planar graph G, the problem of deciding whether G can be contracted to H is fixed-parameter tractable when parameterized by $|V(H)|$.

Our Contribution. We study $\mathcal{H}$-Contraction when $\mathcal{H}$ is the class of planar graphs, and refer to the problem as Planar Contraction. This problem is known to be NP-complete due to a more general result on $\mathcal{H}$-Contraction by Asano and Hirata [2]. We show that the Planar Contraction problem is fixed-parameter tractable when parameterized by k. This result complements the following results on two other graph modification problems related to planar graphs. The problem of deciding whether a given graph can be made planar by using at most k vertex deletions was proved to be fixed-parameter tractable independently by Marx and Schlotter [15], who presented a quadratic-time algorithm for every fixed k, and by Kawarabayashi [12], whose algorithm runs in linear time for every k. Kawarabayashi and Reed [13] showed that deciding whether a graph can be made planar by using at most k edge deletions can also be done in linear time for every fixed k.

Our algorithm for Planar Contraction starts by finding a set S of at most k vertices whose deletion transforms G into a planar graph. Such a set can be found by using either the above-mentioned linear-time algorithm by Kawarabayashi [12] or the quadratic-time algorithm by Marx and Schlotter [15]. The next step of our algorithm is based on the irrelevant vertex technique developed in the graph minors project of Robertson and Seymour [17,19]. We show that if the input graph G has large treewidth, we can find an *edge* whose contraction yields an equivalent, but smaller instance. After repeatedly contracting such irrelevant edges, we invoke Courcelle's Theorem [4] to solve the remaining instance in linear time.

We finish this section by making two remarks that show that we cannot apply the techniques that were used to prove fixed-parameter tractability of the vertex deletion and edge deletion variants of Planar Contraction. First, a crucial observation in the paper of Kawarabayashi and Reed [13] is that any graph that can be made planar by at most k edge deletions must have bounded genus. This property is heavily exploited in the case where the treewidth of the input graph is large. The following example shows that we cannot use this technique in our setting. Take a complete biclique $K_{3,r}$ with partition classes A and B, where $|A| = 3$ and $|B| = r$ for some integer $r \geq 3$. Now make the vertices in A pairwise adjacent and call the resulting graph G_r. Then G_r can be made modified into a planar graph by contracting one of the edges in A. However, the genus of G_r is at least the genus of $K_{3,r}$, which is equal to $\frac{r-2}{2}$ [3].

Second, the problem of deciding whether a graph can be made planar by at most k vertex deletions for some fixed integer k, i.e., k is not part of the input, is called the k-Apex problem. As observed by Kawarabayashi [12] and Marx and Schlotter [15], the class of so-called k-apex graphs (graphs that can be made planar by at most k vertex deletions) is closed under taking minors. This means that the k-Apex problem can be solved in cubic time for any fixed integer k due to deep results by Robertson and Seymour [18]. However, we cannot apply Robertson and Seymour's result on Planar Contraction, because the class of graphs that can be made planar by at most k edge contractions is not closed under taking minors, as the following example shows. Take the complete graph K_5 on 5 vertices. For each edge $e = uv$, add a path P_e from u to v consisting of p new vertices for some integer $p \geq k$. Call the resulting graph G_p^*. Then G_p^* can be made planar by contracting an arbitrary edge of the original K_5. However, if we remove all edges of this K_5, we obtain a minor of G_p^* that is a subdivision of the graph K_5. In order to make this minor planar, we must contract all edges of a path P_e, so we need at least $p + 1 > k$ edge contractions.

2 Preliminaries

Throughout the paper we consider undirected finite graphs that have no loops and no multiple edges. Whenever we consider a graph problem, we use n to denote the number of vertices of the input graph. Let $G = (V, E)$ be a graph and let S be a subset of V. We write $G[S]$ to denote the subgraph of G *induced* by S, i.e., the subgraph of G with vertex set S and edge set $\{uv \mid u, v \in S \text{ with } uv \in E\}$. We write $G - S = G[V \setminus S]$, and for any subgraph H of G, we write $G - H$ to denote $G - V(H)$. We say that two disjoint subsets $U \subseteq V$ and $W \subseteq V$ are *adjacent* if there exist two vertices $u \in U$ and $w \in W$ such that $uw \in E$. Let H be a graph that is not necessarily vertex-disjoint from G. Then $G \cup H$ denotes the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$, and $G \cap H$ denotes the graph with vertex set $V(G) \cap V(H)$ and edge set $E(G) \cap E(H)$.

The *contraction* of edge uv in G removes u and v from G, and replaces them by a new vertex made adjacent to precisely those vertices that were adjacent to u or v in G. A graph H is a *contraction* of G if H can be obtained from G by a sequence of edge contractions. Alternatively, we can define a contraction of G as follows. An *H-witness structure* $\mathcal{W}$ is a partition of $V(G)$ into $|V(H)|$ nonempty sets $W(x)$, one for each $x \in V(H)$, called *H-witness sets*, such that each $W(x)$ induces a connected subgraph of G, and for all $x, y \in V(H)$ with $x \neq y$, the sets $W(x)$ and $W(y)$ are adjacent in G if and only if x and y are adjacent in H.

Clearly, H is a contraction of G if and only if G has an H-witness structure; H can be obtained by contracting each witness set into a single vertex. A *witness edge* is an edge of G whose end-vertices belong to two different witness sets.

Let $\mathcal{W}$ be an H-witness structure of G. For our purposes, we sometimes have to contract edges in G such that the resulting graph does *not* contain H as contraction. In order to do this it is necessary to *destroy* $\mathcal{W}$, i.e., to contract at least one witness edge in $\mathcal{W}$. After all, every edge that is not a witness edge has both its end-vertices in the same witness set of $\mathcal{W}$, which means that contracting such an edge yields an H-witness structure of a contraction of G. Hence, contracting all such non-witness edges transforms G into H itself. Note that if we destroy $\mathcal{W}$ by contracting a witness edge e in $\mathcal{W}$, the obtained graph still has H as a contraction if e was a non-witness edge in some other H-witness structure of G. Hence, in order to obtain a graph that does not have H as a contraction, it is necessary and sufficient to destroy *all* H-witness structures of G.

A *planar graph* G is a graph that can be embedded in the plane, i.e., that can be drawn in the plane so that its edges intersect only at their end-vertices. A graph that is actually drawn in such a way is called a *plane graph*, or an *embedding* of the corresponding planar graph. A plane graph G partitions the rest of the plane into a number of connected regions, called the *faces* of G. Each plane graph has exactly one unbounded face, called the *outer* face; all other faces are called *inner faces*. Let C be a cycle in a plane graph G. Then, by the Jordan Curve Theorem, C divides the plane in exactly two regions: the *outer* region of C, containing the outer face of G, and the *inner* region of C. We say that a vertex u of G lies *inside* C if u is in the inner region of C. Similarly, u lies *outside* C if u is in the outer region of C. The *interior* of C with respect to G, denoted $\text{interior}_G(C)$, is the set of all vertices of G that lie inside C. We also call these vertices *interior* vertices of C. We say that C *separates* the vertices that lie inside C from the vertices that lie outside C. A sequence of mutually vertex-disjoint cycles $C_1, \ldots, C_q$ in a plane graph is called *nested* if there exist disks $\Delta_1, \ldots, \Delta_q$ such that C_i is the boundary of Δ_i for $i = 1, \ldots, q$, and $\Delta_{i+1} \subset \Delta_i$ for $i = 1, \ldots, q-1$. We also refer to such a sequence of nested cycles as *layers*. We say that a vertex u lies *between* two nested cycles C_i and C_j with $i < j$ if u lies in the inner region of C_i and in the outer region of C_j.

A graph G contains a graph H as a *minor* if G can be modified to H by a sequence of edge contractions, edge deletions and vertex deletions. Note that a graph G contains a graph H as a minor if and only if G contains a subgraph that contains H as a contraction. The *subdivision* of an edge $e = uv$ in a graph G removes e from G and replaces it by a new vertex w that is made adjacent to u and v. A *subdivision* of a graph G is a graph obtained from G after performing a sequence of edge subdivisions. In Figure 1, three examples of an *elementary wall* are given. The unique cycle that forms the boundary of the outer face is called the *perimeter* of the wall. A *wall* W of height h is a subdivision of an elementary wall of height h and is well-known to have a unique planar embedding. We also call the facial cycle of W corresponding to the perimeter of the original elementary wall the *perimeter* of W, and we denote this cycle by $P(W)$.

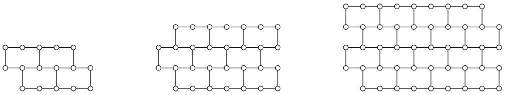


Fig. 1. Elementary walls of height 2, 3, and 4 with perimeters of length 14, 22, and 30, respectively

The $r \times r$ *grid* has all pairs (i, j) for $i, j = 0, 1, \ldots, r-1$ as the vertex set, and two vertices (i, j) and (i', j') are joined by an edge if and only if $|i-i'|+|j-j'| = 1$. The *side length* of an $r \times r$ grid is r. A well-known result of Robertson and Seymour [16] states that, for every integer r, any planar graph with no $r \times r$ grid minor has treewidth at most $6r - 5$. Although it is not known whether a largest grid minor of a planar graph can be computed in polynomial time, there exist several constant-factor approximation algorithms. In our algorithm we will use one by Gu and Tamaki [6]. For any graph G, let $\mathrm{gm}(G)$ be the largest integer r such that G has an $r \times r$ grid as a minor. Gu and Tamaki [6] showed that for every constant $\epsilon > 0$, there exists a constant $c_\epsilon > 3$ such that an $r \times r$ grid minor in a planar graph G can be constructed in time $O(n^{1+\epsilon})$, where $r \geq \mathrm{gm}(G)/c_\epsilon$. Because we can obtain a wall of height $\lfloor r/2 \rfloor$ as a subgraph from an $r \times r$ grid minor by deleting edges and vertices, their result implies the following theorem.

Theorem 1 ([6]). *Let G be a planar graph, and let h^* be the height of a largest wall that appears as a subgraph in G. For every constant $\epsilon > 0$, there exists a constant $c_\epsilon > 3$ such that a wall in G with height at least h^*/c_ϵ can be constructed in time $O(n^{1+\epsilon})$.*

3 Fixed-Parameter Tractability of Planar Contraction

For our algorithm we need the aforementioned result of Kawarabayashi [12].

Theorem 2 ([12]). *For every fixed integer k, it is possible to find in $O(n)$ time a set S of at most k vertices in an n-vertex graph G such that $G - S$ is planar, or conclude that such a set S does not exist.*

We also need three lemmas, whose proofs are left out due to space restrictions.

Lemma 1. *If a graph $G = (V, E)$ can be contracted to a planar graph by using at most k edge contractions, then there exists a set $S \subseteq V$ with $|S| \leq k$ such that $G - S$ is planar.*[1]

When k is fixed, we write k-Planar Contraction instead of Planar Contraction. A seminal result of Courcelle [4] states that on any class of graphs of bounded treewidth, every problem expressible in monadic second-order logic can be solved in time linear in the number of vertices of the graph.

[1] As an aside, we point out that the reverse of this statement is not true. For instance, take a K_5 and subdivide each of its edges $p \geq 3$ times. The resulting graph can be made planar by one vertex deletion, but at least $p-1$ edge contractions are required.

Lemma 2. *For every fixed integer k, the k-*Planar Contraction *problem can be expressed in monadic second-order logic.*

Lemma 3. *Let B be a planar graph that has an embedding with two nested cycles C_1 and C_2, such that C_1 is the boundary of its outer face and C_2 is the boundary of an inner face, and such that there are at least two vertex-disjoint paths that join vertices of C_1 and C_2. Let I be a graph with $B \cap I = C_2$ such that $R = B \cup I$ is planar. Then R has an embedding such that C_1 is the boundary of the outer face.*

We are now ready to present our main theorem, which shows that Planar Contraction is fixed-parameter tractable when parameterized by k. At some places in our proof of Theorem 3 we allow constant factors (independent of k) to be less than optimal in order to make the arguments easier to follow.

Theorem 3. *For every fixed integer k and every constant $\epsilon > 0$, the k-*Planar Contraction *problem can be solved in $O(n^{2+\epsilon})$ time.*

Proof. Let G be a graph on n vertices, and let k be some fixed integer. If G has connected components $L_1, \ldots, L_q$ for some $q \geq 2$, then we solve for every possible tuple $(k_1, \ldots, k_q)$ with $\sum_{i=1}^{q} k_i = k$ the instances $(L_1, k_1), \ldots, (L_q, k_q)$. Hence, we may assume without loss of generality that G is connected. We apply Theorem 2 to decide in $O(n)$ time whether G contains a subset S of at most k vertices such that $G - S$ is planar. If not, then we return no due to Lemma 1. Hence, from now on we assume that we have found such a set S. We write $H = G - S$.

Choose $\epsilon > 0$. We apply Theorem 1 on the graph H to find in $O(n^{1+\epsilon})$ time a subgraph W of H that is a wall with height $h \geq h^*/c_\epsilon$, where h^* denotes the height of a largest wall in H and $c_\epsilon > 3$ is some constant.

Suppose that $h \leq \lceil\sqrt{2k+1}\rceil(12k+10)$. Then $h^* \leq c_\epsilon h \leq c_\epsilon\lceil\sqrt{2k+1}\rceil(12k+10)$, i.e., the height of a largest wall in H is bounded by a constant. Consequently, the treewidth of H is bounded by a constant [16]. Since deleting a vertex from a graph decreases the treewidth by at most 1, the treewidth of G is at most $|S| \leq k$ larger than the treewidth of H. Because k is fixed, this means that the treewidth of G is bounded by a constant as well. Then Lemma 2 tells us that we may apply Courcelle's Theorem [4] to check in $O(n)$ time if G can be modified into a planar graph by using at most k edge contractions.

Now suppose that $h > \lceil\sqrt{2k+1}\rceil(12k+10)$. We consider some fixed planar embedding of H. For convenience, whenever we mention the graph H below, we always refer to this fixed embedding. The wall W is contained in some connected component $\tilde{H}$ of H, and we assume without loss of generality that all other connected components of H lie outside $P(W)$. Inside $P(W)$, we choose $2k+1$ mutually vertex-disjoint subwalls $W_1, \ldots, W_{2k+1}$ of height $12k+8$ that are packed inside W in $\lceil\sqrt{2k+1}\rceil$ rows of $\lceil\sqrt{2k+1}\rceil$ subwalls, such that vertices of distinct subwalls are not adjacent; see Figure 2. Inside each W_i, we choose a subwall W_i' of height $12k+6$ such that the perimeters of W_i and W_i' are vertex-disjoint; see Figure 2 for a depiction of W_i and W_i' in case $k = 0$. By definition, the

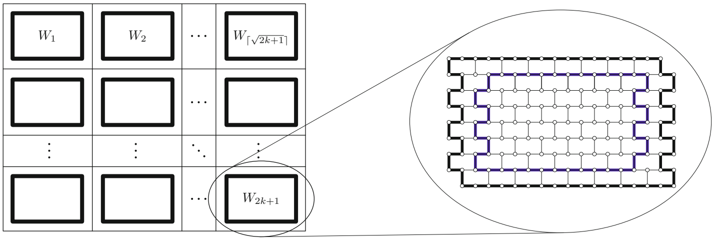


Fig. 2. On the left, a schematic depiction of the wall W with height $h > \lceil\sqrt{2k+1}\rceil(12k+10)$ and the way the subwalls $W_1, \ldots, W_{2k+1}$, each with height $12k+8$, are packed within W. On the right, a more detailed picture of a subwall W_i of height 8 in case $k = 0$. The bold blue edges indicate the perimeter of the smaller subwall W_i' of height 6.

inner region of $P(W_i)$ is the region that contains the vertices of W_i', and the inner region of $P(W_i')$ is the region that contains no vertex of $P(W_i)$. Note that the interiors of $P(W_i)$ and $P(W_i')$ are defined with respect to (the fixed planar embedding of) the graph H. Hence, these interiors may contain vertices of H that do not belong to W, as W is a subgraph of H.

We now consider the graph G. Recall that $H = G - S$, and that G is not necessarily planar. Hence, whenever we speak about the interior of some cycle below, we always refer to the interior of that cycle with respect to the fixed planar embedding of H. For $i = 1, \ldots, 2k+1$, let $S_i \subseteq S$ be the subset of vertices of S that are adjacent to an interior vertex of $P(W_i')$. Observe that the sets S_i are not necessarily disjoint, since a vertex of S might be adjacent to interior vertices of $P(W_i')$ for several values of i. Also note that no vertex of S belongs to W, since W is a wall in the graph $H = G - S$. We can construct the sets S_i in $O(n)$ time, because the number of edges of G is $O(n)$. The latter can be seen as follows. The number of edges in G is equal to the sum of the number of edges of H, the number of edges between H and S, and the number of edges of $G[S]$. Because H is planar, the number of edges of H is at most $5|V(H)| \leq 5n$. Hence, the number of edges of G is at most $5n + kn + \frac{1}{2}k(k-1) = O(n)$ for fixed k.

We say that a set S_i is of *type 1* if S_i is non-empty and if every vertex $y \in S_i$ also belongs to some set S_j for $j \neq i$, i.e., every vertex $y \in S_i$ is adjacent to some vertex z that lies inside $P(W_j')$ for some $j \neq i$; see Figure 3 for an illustration. Otherwise we say that S_i is of *type 2*. We can check in $O(n)$ time how many sets S_i are of type 1. We claim the following.

Claim 1. If there are at least $k+1$ sets S_i of type 1, then (G, k) is a no-instance.

We prove Claim 1 as follows. Suppose that there exist $\ell \geq k+1$ sets S_i of type 1, say these sets are $S_1, \ldots, S_\ell$. Then for each $i = 1, \ldots, \ell$ we can define a K_5-witness structure $\mathcal{X}_i$ of a subgraph of G as follows. We divide the perimeter of W_i into three connected non-empty parts in the way illustrated in Figure 3. The vertices of each part will form a separate witness set of $\mathcal{X}_i$; let us call these witness sets X_i^1, X_i^2, X_i^3. Let H_i' be the subgraph of H induced by the vertices

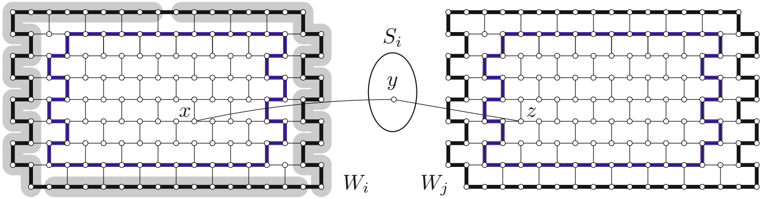


Fig. 3. Two subwalls W_i and W_j, where the bold blue edges indicate the perimeters $P(W_i')$ and $P(W_j')$ of the smaller subwalls W_i' and W_j'. Vertex y is in S_i, since it is adjacent to an interior vertex x of $P(W_i')$. If, for every $y \in S_i$, there is an edge between y and an interior vertex z of $P(W_j')$ for some $j \neq i$, then S_i is of type 1. The three shaded areas indicate how the perimeter of W_i is divided into three non-empty parts, each forming a separate witness set of a K_5-witness structure $\mathcal{X}_i$ of a subgraph of G.

that lie inside $P(W_i)$ in H. The fourth set X_i^4 of the witness structure $\mathcal{X}_i$ consists of all the vertices of the connected component of H_i' that contains W_i'. Let G_i' be the graph obtained from G by deleting all the vertices of $P(W_i)$ and all the vertices that lie inside $P(W_i)$ in H, i.e., $G_i' = G - P(W_i) - \text{interior}_H(P(W_i))$. Let D be the connected component of G_i' that contains the perimeter $P(W)$ of the large wall W. It is clear that D contains all vertices that are on or inside $P(W_j)$ for every $j \neq i$. Hence, due to the assumption that S_i is of type 1, all vertices of S_i also belong to D. The fifth set X_i^5 is defined to be the vertex set of D. Let us argue why these five sets form a K_5-witness structure of a subgraph of G.

It is clear that each of the sets X_i^1, X_i^2, X_i^3 is connected, and that they are pairwise adjacent. The set X_i^4 is connected by definition. The choice of the subwall W_i' within W_i ensures that X_i^4 is adjacent to each of the sets X_i^1, X_i^2, X_i^3. Let us consider the set X_i^5. By definition, X_i^5 is connected. Since X_i^5 contains the perimeter $P(W)$ of the large wall W and W_i lies inside $P(W)$, set X_i^5 is adjacent to X_i^1, X_i^2 and X_i^3. Since S_i is of type 1 and hence non-empty, there is a vertex $y \in S_i$ that is adjacent to a vertex x that lies inside $P(W_i')$ by the definition of S_i. We already argued that X_i^5 contains all vertices of S_i, so $y \in X_i^5$. Recall that $\tilde{H}$ is the unique connected component of H that contains the wall W, and that all connected components of H other than $\tilde{H}$ were assumed to lie outside $P(W)$ in H. Because x lies inside $P(W_i')$, this means that x is in the connected component of H_i' that contains W_i', implying that $x \in X_i^4$. Consequently, the edge between x and y ensures the adjacency between X_i^4 and X_i^5.

We now consider the ℓ different K_5-witness structures $\mathcal{X}_i$ of subgraphs of G defined in the way described above, one for each $i \in \{1, \ldots, \ell\}$. Let us see how such a K_5-witness structure $\mathcal{X}_i$ can be destroyed by using edge contractions only. Denote by E_i the set of edges of G incident with the vertices of $X_i^1 \cup \cdots \cup X_i^4$ for $i = 1, \ldots, \ell$. We can only destroy a witness structure $\mathcal{X}_i$ by edge contractions if we contract the edges of at least one path that has its endvertices in different witness sets of $\mathcal{X}_i$ and its inner vertices (in case these exist) not belonging to any witness set of $\mathcal{X}_i$. Clearly, such a path always contains an edge of E_i. Hence, in order to destroy $\mathcal{X}_i$, we have to contract at least one edge of E_i. Because the sets $E_1, \dots, E_\ell$ are pairwise disjoint by the construction of the witness structures $\mathcal{X}_i$, we must use at least $\ell \geq k+1$ edge contractions in order to make G planar. Hence, G is a no-instance. This proves Claim 1.

Due to Claim 1, we are done if there are at least $k+1$ sets S_i of type 1. Note that every step in our algorithm so far took $O(n^{1+\epsilon})$ time, as desired. Suppose that we found at most k sets S_i of type 1. Because the total number of sets S_i is $2k+1$, this means that there are at least $k+1$ sets S_i of type 2. Let S_i be a set of type 2. If S_i is non-empty, then S_i contains a vertex x that is not adjacent to an interior vertex of $P(W'_j)$ for any $j \neq i$, as otherwise S_i would be of type 1. Consequently, S_i is the only set of type 2 that contains x. Since $|S| \leq k$ and there are at least $k+1$ sets of type 2, at least one of them must be empty. Without loss of generality, we assume from now on that $S_1 = \emptyset$.

We will now exploit the property that $S_1 = \emptyset$, i.e., that none of the vertices in the interior of $P(W'_1)$ is adjacent to any vertex of S. We define a *triple layer* as the perimeter of a wall with the perimeters of its two largest proper subwalls inside, such that the three perimeters are mutually vertex-disjoint, and the *middle* perimeter is adjacent to the *outer* and *inner* perimeter. We define a sequence of nested triple layers in the same way as we defined a sequence of layers in Section 2. Because W'_1 has height $12k+6$, there exist two adjacent vertices u and v inside $P(W'_1)$, such that u and v are separated from the vertices outside $P(W'_1)$ by $2k+1$ nested triple layers $L_1, \dots, L_{2k+1}$, i.e., u and v lie inside the inner perimeter of triple layer L_{2k+1}.

Let G' be the graph obtained from G after contracting uv. The following claim shows that uv is an "irrelevant" edge, i.e., that uv may be contracted without loss of generality.

Claim 2. (G, k) is a yes-instance if and only if (G', k) is a yes-instance.

We prove Claim 2 as follows. First suppose that (G, k) is a yes-instance. This means that G can be modified into a planar graph F by at most k edge contractions. Let $E' \subseteq E(G)$ be a set of at most k edges whose contraction modifies G into F. Observe that we can contract the edges in E' in any order to obtain F from G. If $uv \in E'$, then we can first contract uv to obtain the graph G', and then contract the other edges in E' to modify G' into the planar graph F. If $uv \notin E'$, then we first contract the edges in E' to modify G into F, and then contract the edge uv. This leads to a graph F'. Since planar graphs are closed under edge contractions, F' is planar. Moreover, F' can also be obtained from G' by contracting the edges in E'. We conclude that (G', k) is a yes-instance.

Now suppose that (G', k) is a yes-instance. This means that G' can be modified into a planar graph F' by at most k edge contractions. Let $E' \subseteq E(G')$ be a set of at most k edges whose contraction modifies G' into F'. Let F be the graph obtained from G by contracting all the edges of E'. We will show that F is planar as well.

Recall that $S_1 = \emptyset$, and that we defined $2k+1$ triple layers $L_1, \dots, L_{2k+1}$ inside $P(W'_1)$. Let Q_i, Q'_i, and Q''_i denote the three perimeters in H that form the triple layer L_i for $i = 1, \ldots, 2k+1$, where Q_i is the outer perimeter, Q_i' the middle perimeter, and Q_i'' the inner perimeter. Let Y_i be the set of all vertices of H that are in $Q_i \cup Q_i' \cup Q_i''$ or that lie in the intersection of the inner region of Q_i and the outer region of Q_i'', i.e., Y_i is the set of vertices that lie on or "in between" the perimeters Q_i and Q_i'' in H. Because we applied at most k edge contractions in G', there exists a set Y_i, for some $1 \leq i \leq 2k+1$, such that none of its vertices is incident with an edge in E'. This means that L_i is a triple layer in F' as well. We consider a planar embedding of F', in which Q_i'' is in the inner region of Q_i', and Q_i' is in the inner region of Q_i; for convenience, we will denote this planar embedding by F' as well.

We will now explain how to apply Lemma 3. We define C_1 and C_2 to be the perimeters Q_i' and Q_i'', respectively. We define B as the subgraph of F' induced by the vertices that either are in $Q_i' \cup Q_i''$ or lie between Q_i' and Q_i'' in F'. Here, we assume that B is connected, as we can always place connected components of B that do not contain vertices from $Q_i' \cup Q_i''$ outside Q_i'. Because Q_i' and Q_i'' are perimeters of subwalls in H, and Q_i'' is contained inside Q_i', there exist at least two vertex-disjoint paths P_1, P_2 in H joining Q_i' and Q_i'' using vertices of Y_i only. Because none of the vertices in Y_i is incident with an edge in E', the two paths P_1, P_2 are also vertex-disjoint in F', and consequently in B.

We now construct the graph I. Because F' is a contraction of G', and G' is a contraction of G, we find that F' is a contraction of G. Let $\mathcal{W}$ be an F'-witness structure of G corresponding to contracting exactly the edges of $E' \cup \{uv\}$ in G. Then we define I to be the subgraph of G induced by the union of the vertices of all the witness sets $W(x)$ with x on or inside Q_i'' in F'. Just as we may assume that B is connected, we may also assume that the subgraph of F' induced by the vertices that lie on or inside Q_i'' is connected. Because witness sets are connected by definition, we then find that I is connected.

Because the edge uv is contracted when G is transformed into F', u and v belong to the same witness set of $\mathcal{W}$. Let x^* be the vertex of F', such that u and v are in the witness set $W(x^*)$. Recall that all the vertices of Q_i'' and the vertices u and v belong to the wall W_1'. Since u and v lie inside Q_i'' in H and walls have a unique plane embedding, x^* lies inside Q_i'' in F'. Hence, u and v are vertices of I. Also recall that none of the vertices of Y_i, and none of the vertices of Q_i'' in particular, is incident with an edge of E'. Hence, the vertices of Q_i'' correspond to witness sets of $\mathcal{W}$ that are singletons, i.e., that have cardinality 1. This means that we can identify each vertex of Q_i'' in F' with the unique vertex of G in the corresponding witness set. Hence, we obtain that $B \cap I = Q_i'' = C_2$.

We now prove that $R = B \cup I$ is planar. For doing this, we first prove that B contains no vertex x with $W(x) \cap S \neq \emptyset$, and that I contains no vertex from S. To see that B contains no vertex x with $W(x) \cap S \neq \emptyset$, assume that x is a vertex of B and s is a vertex of S with $s \in W(x)$. Recall that no vertex from Q_i' is incident with an edge in E'. Hence, we can identify each vertex in Q_i' in F' with the unique vertex of the corresponding witness set, just as we did earlier with the vertices of Q_i''. Because $s \in W(x)$, this means that x is not in Q_i'. Because B is connected, we find that F' contains a path from x to a vertex y in Q_i' that contains no vertex from Q_i. Note that since y is in Q_i', y is a vertex in G as well. Because $W(x)$ induces a connected subgraph of G by definition, this path can be transformed into a path in G from s to y that does not contain a vertex from Q_i. This is not possible, because $S_1 = \emptyset$ implies that every path in G from s to y must go through Q_i.

We now show that I contains no vertex from S. In order to obtain a contradiction, assume that I contains a vertex $s \in S$. Because I is connected, this means that G contains a path from s to a vertex in Q_i'' that contains no vertex from Q_i (and also no vertex from Q_i'). This is not possible, because $S_1 = \emptyset$.

Let R' be the subgraph of G induced by the vertices in the sets $W(x)$ with $x \in V(B)$ and the vertices of I. Since we proved that R' contains no vertices from S, R' is a subgraph of H. Consequently, R' is planar because H is planar. As a result, R is planar because R can be obtained from R' by contracting all edges in every set $W(x)$ with $x \in V(B)$, and planar graphs are closed under edge contractions. As we have shown that $R = B \cup I$ is planar, we are now ready to apply Lemma 3. This lemma tells us that R has an embedding $\mathcal{R}$, such that $Q_i' = C_1$ is the boundary of the outer face. Now consider the embedding that we obtain from the (plane) graph F' by removing all vertices that lie inside Q_i'. We combine this embedding with $\mathcal{R}$ to obtain a plane embedding of a graph F^*. We can obtain F from F^* by contracting all edges in E' that are incident to a vertex in I; recall that u and v are both in I and that uv is not an edge of E'. Because planar graphs are closed under edge contractions, this means that F is planar. This completes the proof of Claim 2.

We can find the irrelevant edge uv mentioned just above Claim 2 in $O(n)$ time. Since all other steps took $O(n^{1+\epsilon})$ time, we used $O(n^{1+\epsilon})$ time so far. After finding the edge uv, we contract it and continue with the smaller graph G'. Because removing S will make G' planar as well, we can keep S instead of applying Theorem 2 again. Hence, we apply Theorem 2 only once. Because G has n vertices, and every iteration reduces the number of vertices by exactly one, the total running time of our algorithm is $O(n^{2+\epsilon})$. This completes the proof. □

4 Conclusions

We proved that Planar Contraction is fixed-parameter tractable when parameterized by k. Very recently, Abello et al. [1] independently showed that the closely related problem that is to test whether a given graph can be made planar by contracting the edges of at most k mutually vertex-disjoint subgraphs, each of which of size at most ℓ, can be solved in quadratic time for any fixed k and $\ell \geq 2$. Their algorithm can easily be modified to show that k-Planar Contraction can be solved in quadratic time for any fixed k (just as we can modify our algorithm to solve their problem).

A natural direction for future work is to consider the class $\mathcal{H}$ that consists of all H-minor free graphs for some graph H and to determine the parameterized complexity of $\mathcal{H}$-Contraction for such graph classes. Our proof techniques rely on the fact that we must contract to a planar graph, and as such they cannot be used directly for this variant. Hence, we pose this problem as an open problem.

References

1. Abello, J., Klavík, P., Kratochvíl, J., Vyskočil, T.: Matching and ℓ-subgraph contractibility to planar graphs. Manuscript, arXiv:1204.6070 (2012)
2. Asano, T., Hirata, T.: Edge-contraction problems. Journal of Computer and System Sciences 26, 197–208 (1983)
3. Bouchet, A.: Orientable and nonorientable genus of the complete bipartite graph. Journal of Combinatorial Theory, Series B 24, 24–33 (1978)
4. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Information and Computation 85, 12–75 (1990)
5. Golovach, P.A., Kamiński, M., Paulusma, D., Thilikos, D.M.: Increasing the Minimum Degree of a Graph by Contractions. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 67–79. Springer, Heidelberg (2012)
6. Gu, Q.-P., Tamaki, H.: Constant-factor approximations of branch-decomposition and largest grid minor of planar graphs in $O(n^{1+\varepsilon})$ time. Theoretical Computer Science 412, 4100–4109 (2011)
7. Heggernes, P., van 't Hof, P., Lévêque, B., Paul, C.: Contracting chordal graphs and bipartite graphs to paths and trees. In: Proc. LAGOS 2011. ENDM, vol. 37, pp. 87–92 (2011)
8. Heggernes, P., van 't Hof, P., Lévêque, B., Lokshtanov, D., Paul, C.: Contracting Graphs to Paths and Trees. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 55–66. Springer, Heidelberg (2012)
9. Heggernes, P., van 't Hof, P., Lokshtanov, D., Paul, C.: Obtaining a bipartite graph by contracting few edges. Algorithmica (to appear), doi: 10.1007/s00453-012-9670-2
10. Kamiński, M., Paulusma, D., Thilikos, D.M.: Contractions of Planar Graphs in Polynomial Time. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 122–133. Springer, Heidelberg (2010)
11. Kamiński, M., Thilikos, D.M.: Contraction checking in graphs on surfaces. In: Proc. STACS, pp. 182–193 (2012)
12. Kawarabayashi, K.: Planarity allowing few error vertices in linear time. In: Proc. FOCS, pp. 639–648 (2009)
13. Kawarabayashi, K., Reed, B.A.: Computing crossing number in linear time. In: Proc. STOC, pp. 382–390 (2007)
14. Marx, D., O'Sullivan, B., Razgon, I.: Finding small separators in linear time via treewidth reduction. Manuscript, arXiv:1110.4765 (2012)
15. Marx, D., Schlotter, I.: Obtaining a planar graph by vertex deletion. Algorithmica 62, 807–822 (2012)
16. Robertson, N., Seymour, P.D.: Quickly excluding a planar graph. Journal of Combinatorial Theory, Series B 62, 323–348 (1994)
17. Robertson, N., Seymour, P.D.: Graph minors XIII: The disjoint paths problem. Journal of Combinatorial Theory, Series B 63, 65–110 (1995)
18. Robertson, N., Seymour, P.D.: Graph minors XX: Wagner's conjecture. Journal of Combinatorial Theory, Series B 92, 325–357 (2004)
19. Robertson, N., Seymour, P.D.: Graph minors XXII: Irrelevant vertices in linkage problems. Journal of Combinatorial Theory, Series B 102, 530–563 (2012)

Light Spanners in Bounded Pathwidth Graphs

Michelangelo Grigni and Hao-Hsiang Hung

Dept. of Math & CS, Emory University
{mic,hhung2}@mathcs.emory.edu

Abstract. Given an edge-weighted graph G and $\epsilon > 0$, a $(1+\epsilon)$-spanner is a spanning subgraph G' whose shortest path distances approximate those of G within a factor of $1+\epsilon$. For G from certain graph families (such as bounded genus graphs and apex graphs), we know that *light* spanners exist. That is, we can compute a $(1+\epsilon)$-spanner G' with total edge weight at most a constant times the weight of a minimum spanning tree. This constant may depend on ϵ and the graph family, but not on the particular graph G nor on the edge weighting. The existence of light spanners is essential in the design of approximation schemes for the metric TSP (the traveling salesman problem) and similar graph-metric problems.

In this paper we make some progress towards the conjecture that light spanners exist for every minor-closed graph family: we show that light spanners exist for graphs with bounded pathwidth, and they are computed by a greedy algorithm. We do this via the intermediate construction of light *monotone* spanning trees in such graphs.

1 Introduction

1.1 Light Spanners

Suppose G is a connected undirected graph where each edge e has length (or weight) $w(e) \geq 0$. Let $d_G(u,v)$ denote the length of the shortest path between vertices u and v. Suppose G' is a spanning subgraph of G, where each edge of G' inherits its weight from G; evidently $d_G(u,v) \leq d_{G'}(u,v)$. Fix $\epsilon > 0$. If $d_{G'}(u,v) \leq (1+\epsilon) \cdot d_G(u,v)$ (for all u, v), then we say that G' is a $(1+\epsilon)$-*spanner* of G. In other words, the metric $d_{G'}$ closely approximates the metric d_G.

Let $w(G')$ denote the total edge weight of G', and let $\mathrm{MST}(G)$ denote the minimum weight of a spanning tree in G. We are interested in conditions on G that guarantee the existence of a $(1+\epsilon)$-spanner G' with bounded $w(G')/\mathrm{MST}(G)$. Suppose $\mathcal{G}$ is a family of undirected graphs. We say $\mathcal{G}$ *has light spanners* if the following holds: for every $\epsilon > 0$ there is a bound $f(\epsilon)$, so that for any edge-weighted G from $\mathcal{G}$, G has a $(1+\epsilon)$-spanner G' with $w(G') \leq f(\epsilon) \cdot \mathrm{MST}(G)$. Less formally, we say that G' is a *light spanner* for G. Note $f(\epsilon)$ depends on ϵ and $\mathcal{G}$, but not on G or w.

We know that if a graph family has unbounded clique minors, then it does not have light spanners; just consider a clique with uniform edge weights. We conjecture the converse [7]:

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 467–477, 2012.

Conjecture 1. Any graph family with a forbidden minor has light spanners.

Our pursuit of this conjecture is guided by the Robertson-Seymour theory [12], which characterizes minor-closed graph families using four elements: bounded genus graphs, apices, vortices, and repeated clique-sums. We already know that if G has bounded genus or is an apex graph, then it has light spanners [7,8]. In particular vortices are bounded pathwidth subgraphs, stitched inside the faces of a bounded genus graph.

1.2 Motivation

Conjecture 1 seems like a natural question, and its proof would address the "main difficulty" discussed in the concluding remarks of Demaine *et al.* [6], in the general context of approximation algorithms on weighted graphs. As a specific motivating problem, we review some results on the *metric TSP*, the Traveling Salesman Problem with triangle inequality. (For some other problems, see [3,4].)

We are given an edge-weighted graph G, and we seek a cyclic order of its vertices with minimum total distance as measured by d_G. Equivalently, we want a minimum weight cyclic tour in G visiting each vertex at least once. Let $\mathrm{OPT}(G)$ denote the minimum tour weight; it is well known that $\mathrm{MST}(G) \leq \mathrm{OPT}(G) \leq 2 \cdot \mathrm{MST}(G)$. We seek an approximation scheme: an algorithm which takes as inputs the weighted graph G and $\epsilon > 0$, and which outputs a tour with weight at most $(1 + \epsilon) \cdot \mathrm{OPT}(G)$.

The problem is MAX SNP-hard [11], so we consider approximation schemes where the input graph G is restricted to some graph family $\mathcal{G}$ (e.g., planar graphs). We would like a PTAS (an approximation scheme running in time $O(n^{g(\epsilon)})$, for some function g), or better yet an EPTAS (an approximation scheme running in time $O(g(\epsilon) \cdot n^c)$, where the constant c is independent of ϵ).

Suppose $\mathcal{G}$ is a graph family, and that for any $G \in \mathcal{G}$ we can compute a $(1+\epsilon)$-spanner G' with $w(G') \leq f(\epsilon) \cdot \mathrm{MST}(G)$. Then we may attempt to design a PTAS (or an EPTAS) for the metric TSP on $\mathcal{G}$, as follows:

1. On input G and ϵ, first compute G', a $(1 + \epsilon/2)$-spanner of G, with weight at most $f(\epsilon/2) \cdot \mathrm{MST}(G)$.
2. Choose $\delta = (\epsilon/2)/f(\epsilon/2)$. Apply some algorithm finding a tour in G' with cost at most $\mathrm{OPT}(G') + \delta \cdot w(G')$.
3. Return the tour, with cost at most $(1+\epsilon/2)\cdot\mathrm{OPT}(G)+\delta\cdot(f(\epsilon/2)\cdot\mathrm{MST}(G)) \leq (1 + \epsilon) \cdot \mathrm{OPT}(G)$. (For other metric optimization problems, it may be less trivial to lift a solution from G' back to G.)

Step 2 looks like the original problem, except now we allow an error term proportional to $w(G')$ instead of $\mathrm{OPT}(G')$. This approach has already succeeded for planar graphs [2,9] and bounded genus graphs [6,7].

A recent result of Demaine *et al.* [5, Thm. 2] implies a PTAS for metric TSP when $\mathcal{G}$ is *any* graph class with a fixed forbidden minor. Since we do not know that $\mathcal{G}$ has light spanners, for step 1 they substitute a looser result [8], finding a $(1 + \epsilon)$-spanner G' with weight $O((\log n)/\epsilon) \cdot \mathrm{MST}(G)$ (the hidden constant depending on $\mathcal{G}$). In step 2 their algorithm runs in time $2^{O(1/\delta+\log n)}$.

Their $1/\delta$ is $O(w(G')/(\mathrm{MST}(G) \cdot \epsilon)) = O((\log n)/\epsilon^2)$, so their running time is $n^{O(1/\epsilon^2)}$. If we could compute light spanners for $\mathcal{G}$, then δ would improve to something independent of n, and this would yield an EPTAS for metric TSP on $\mathcal{G}$. (Or alternatively, it would yield an approximation scheme allowing ϵ to slowly approach zero, as long as $1/\delta$ stays $O(\log n)$.)

1.3 Our Work

In this paper we make some progress towards Conjecture 1: we show that light spanners exist for bounded pathwidth graphs.

Theorem 1. *Bounded pathwidth graphs have light spanners, computable by a greedy algorithm.*

We prove this in Section 3. This result is not algorithmically interesting by itself, since metric TSP (and many other problems) is exactly solvable in polynomial time when G has bounded pathwidth, or even bounded treewidth. Rather, we regard this as progress towards the conjecture, and towards an EPTAS for metric TSP (and similar problems) on graphs with forbidden minors. See Section 4 for some further remarks.

2 Preliminaries

2.1 Charging Schemes

In order to exhibit light spanners in a weight-independent way, we use *charging schemes* [8]. (We use the notion called "0-schemes" in [8], not the more general "ϵ-schemes" required for apex graphs.) Suppose each edge of graph G can hold some quantity of *charge*, initially zero. A *detour* is an edge $e \in E$ and a path P such that $e + P$ is a simple cycle in G. For each detour (e, P) we introduce a variable $x_{(e,P)} \geq 0$. Each $x_{(e,P)}$ describes a *charging move*: it subtracts $x_{(e,P)}$ units of charge from edge e, and adds $x_{(e,P)}$ units of charge to each edge of P. When $x_{(e,P)} > 0$, we say "e charges P".

Given graph G, a spanning tree T, and a number v, a *charging scheme from G to T of value v* is an assignment of nonnegative values to the $x_{(e,P)}$ variables (i.e., a fractional sum of detours) meeting the three conditions listed below. Here $\mathrm{out}(e)$ denotes the total charge subtracted from edge e, $\mathrm{in}(e)$ denotes the total charge added to e (as part of various detour paths), and $\mathrm{net}(e) = \mathrm{in}(e) - \mathrm{out}(e)$ is the total charge on e after all the moves are done:

$$\begin{array}{lrl} (1) & \mathrm{out}(e) \geq 1 & \text{for all } e \in G - T, \\ (2) & \mathrm{net}(e) \leq 0 & \text{for all } e \in G - T, \\ (3) & \mathrm{net}(e) \leq v & \text{for all } e \in T. \end{array}$$

Note "$e \in G - T$" means e is an edge of G but not T. As we'll see in Theorem 2, charging schemes imply light spanners.

Definition 1. *An* acyclic scheme *is a charging scheme with two additional conditions:*

(4) *If edge e charges some path, then $e \in G - T$.*
(5) *There is an ordering of the edges such that whenever edge e_1 charges a path containing edge e_2, e_1 precedes e_2.*

For example, planar graphs have integral acyclic schemes of value $v = 2$ [1].

Definition 2. *Suppose we have detours (e_1, P_1) and (e_2, P_2), with $e_2 \in P_1$ and $e_1 \notin P_2$. Their* shortcut *is the detour (e_1, P'), where P' is the path derived from P_1 by replacing e_2 with P_2, and then reducing that walk to a simple path.*

Lemma 1. *Suppose we have an acyclic scheme of value v from G to T, and an edge e in $G - T$. Then there is an acyclic scheme of value v from $G - e$ to T.*

Proof. Let $e_2 = e$. While $\text{in}(e_2)$ is positive, we find some e_1 charging a path P_1 containing e_2. Since $\text{net}(e_2) \leq 0$, e_2 also charges some path P_2. P_2 cannot contain e_1, since the scheme is acyclic. Let $\alpha = \min(x_{(e_1,P_1)}, x_{(e_2,P_2)})$. Now reduce both $x_{(e_1,P_1)}$ and $x_{(e_2,P_2)}$ by α, and increase $x_{(e_1,P')}$ (their shortcut) by α. After this change all the conditions are still satisfied, except possibly for condition (1) at e_2. Repeat until $\text{in}(e_2)$ reaches zero. Finally remove e_2 and any remaining charges out of e_2. □

Theorem 2. *Suppose G is a graph with spanning tree T, and we have an acyclic scheme from G to T of value v. Then for any $\epsilon > 0$, and for any non-negative edge-weighting w on G, a simple greedy algorithm finds a $(1 + \epsilon)$-spanner G' in G containing T, with total weight $w(G') \leq (1 + v/\epsilon) \cdot w(T)$.*

We use the following greedy algorithm of Althöfer *et al.* [1], modified to force the edges of T into G':

```
Spanner($G$, $T$, $1 + \epsilon$):
    $G' = T$
    for each edge $e \in G - T$, in non-decreasing $w(e)$ order
        if $(1 + \epsilon) \cdot w(e) < d_{G'}(e)$ then
            add edge $e$ to $G'$
    return $G'$
```

The proof of Theorem 2 is a variant of previous arguments by LP duality [7,8], for completeness we sketch it here.

Proof. Since G' is computed by the greedy algorithm, it is clearly a $(1 + \epsilon)$-spanner of G containing T; the issue is to bound its weight $w(G')$. By Lemma 1 we have an acyclic scheme from G' to T of value v.

Consider a detour (e, P) in G' with $e \notin T$. We claim $(1 + \epsilon) \cdot w(e) < w(P)$ (to see this, compare e with the last edge inserted by the algorithm on the cycle $e + P$). Multiply through by $x_{(e,P)}$ and we have this:

$$x_{(e,P)} \cdot \epsilon \cdot w(e) \leq x_{(e,P)} \cdot (w(P) - w(e))$$

When $e \in T$ this is still valid, since $x_{(e,P)} = 0$. Now sum over all detours (e, P):

$$\begin{aligned} \sum_{(e,P)} x_{(e,P)} \cdot \epsilon \cdot w(e) &\leq \sum_{(e,P)} x_{(e,P)} \cdot (w(P) - w(e)) \\ \epsilon \cdot \sum_{e \in G'} w(e) \cdot \mathrm{out}(e) &\leq \sum_{e \in G'} w(e) \cdot \mathrm{net}(e) \\ \epsilon \cdot w(G' - T) &\leq v \cdot w(T) \end{aligned}$$

So $w(G') = w(T) + w(G' - T) \leq (1 + v/\epsilon) \cdot w(T)$. □

2.2 Bounded Pathwidth and Monotone Trees

Suppose $G = (V, E)$ is a graph, P is a path (disjoint from G), and $\mathcal{B} = (B_i)_{i \in P}$ is a collection of subsets of V (bags) indexed by vertices i in P. We call the pair $(P, \mathcal{B})$ a *path decomposition* of G if the following conditions hold: (1) $\bigcup_{i \in P} B_i = V$; (2) for every edge $\{u, v\} \in E$, there is at least one bag B_i with $\{u, v\} \subseteq B_i$; (3) for every $v \in V$, $\{i : v \in B_i\}$ is connected (an interval) in P. The *pathwidth* of the decomposition is the maximum bag size minus one, and the pathwidth of G is the minimum pathwidth of any path decomposition of G.

Given $(P, \mathcal{B})$, we may lay out P on the line, and regard G as a subgraph of an interval graph. That is, for each vertex v we have a line interval I_v (corresponding to an interval in P), and we have $I_u \bigcap I_v \neq \emptyset$ whenever $\{u, v\} \in E$, and at most $k+1$ intervals overlap at any point of the line. For convenience we may eliminate ties via small perturbation, so that all the interval endpoints are distinct. In particular, let left(v) denote the leftmost point of I_v.

Suppose T is a rooted tree in G. We say T is a *monotone tree* if for every vertex v in T with parent p, we have left$(p) <$ left(v). When T is a path rooted at an endpoint, we say it is a *monotone path*. In particular if T is a monotone spanning tree in G, then from any vertex v, we can find a monotone path in T from v to the root of T (the vertex with the leftmost interval). For this process, it is convenient to imagine that edges connect intervals at their leftmost intersection point.

3 Main Argument

We are given $\epsilon > 0$, a connected edge-weighted graph G with n vertices, and an interval representation $\{I_v\}$ of G with pathwidth k. We want to find a $(1+\epsilon)$-spanner G' in G of low weight. First we apply some reductions to simplify G:

Nice Decomposition. We may assume that each pair of consecutive bags (as vertex sets) differ by only one vertex. This can be enforced by an argument similar to the construction of *nice* tree-decompositions [10]: if two consecutive bags differ on $m \geq 2$ vertices, we introduce $m - 1$ intermediate bags, in such a way that each pair differs on only one vertex, and we do not increase the maximum bag size. This does not modify G at all.

Bounded Degree Assumption. We may assume each vertex appears in $O(k)$ bags, and so the maxdegree of G is $O(k)$. To enforce this, we copy the bags of G from left to right. After each group of k original bags, ending with a bag B, we insert $|B|$ "replacer" bags, each of which replaces one vertex $v \in B$ with a copy v', connected to v by an edge of length zero. This ends with a bag B', where every vertex $v \in B$ has been replaced by a copy $v' \in B'$. See Figure 1. We continue in this way (using the copies in place of the originals) across the entire path decomposition. If we aren't careful the pathwidth may increase by one, but this does not matter for our asymptotic results. The original graph is obtained by contracting a set S of weight-zero edges in the modified graph. So given a spanner G' in this modified graph, we may contract S in $G' \cup S$ to recover a spanner (of no greater weight) in the original.

Completion Assumption. We may assume that G is *completed*; that is, it contains all edges allowed by its overlapping intervals. In other words: we have a clique in each bag, G is an interval graph. For each absent edge $e = \{u, v\}$, we simply add it with weight $w(e)$ equal to the shortest path length $d_G(u, v)$. This does not change d_G at all. Given a spanner G' in the completed graph, we recover a spanner in the original graph by replacing each completion edge by the corresponding shortest path.

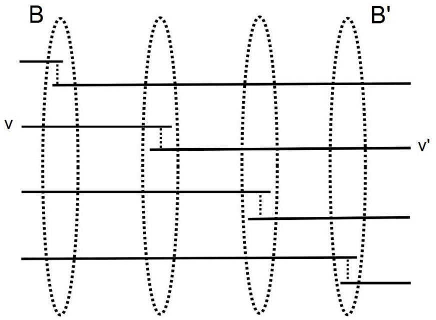


Fig. 1. Each vertex v in bag B is replaced by v' in bag B'

Proof (of Theorem 1). We assume all the above reductions have been applied: the input graph G is a connected edge-weighted interval graph of width k, each bag in its path decomposition introduces at most one vertex, and each vertex of G has degree $O(k)$.

By Lemma 2 (below), we compute a monotone spanning tree T with $w(T) = O(k^2) \cdot \mathrm{MST}(G)$. By Lemma 3 (below), we exhibit an acyclic charging scheme from G to T of value $v = O(k)$. Finally we apply the greedy algorithm, which computes a $(1+\epsilon)$-spanner G'. By Theorem 2, $w(G') \leq (1 + v/\epsilon) \cdot w(T) = O(k^3/\epsilon) \cdot \mathrm{MST}(G)$. □

Lemma 2. *Given G as above, it contains a monotone spanning tree T with $w(T) \leq O(k^2) \cdot MST(G)$.*

Proof. Choose a minimum spanning tree T^*, so $w(T^*) = \mathrm{MST}(G)$. Let I_l and I_r be the leftmost and rightmost intervals. Let P_1 be a shortest path from I_l to I_r; since G is completed, we may assume P_1 is monotone, as in Figure 2. Note $w(P_1) \leq w(T^*)$.

Consider the components ${T_1}^*, {T_2}^*, ..., {T_m}^*$ of $T^* - V(P_1)$. Let e_i be an edge connecting the leftmost point of T_i^* to a vertex of P_1 (it exists by completion). For each ${T_i}^*$, we recursively compute a monotone spanning tree T_i of $G[V({T_i}^*)]$. Finally, $T = P_1 \cup \bigcup_i (T_i \cup e_i)$.

It is clear that T is monotone, but we must account for the total weight of $w(T)$. For each component ${T_i}^*$, let f_i be an edge of T^* connecting ${T_i}^*$ to P_1 (there must be at least one). By triangle inequality, we see $w(e_i)$ is at most $w({T_i}^*) + w(f_i) + w(P_{1,i})$, where $P_{1,i}$ is a subpath of P_1 from the endpoint of e_i to the endpoint of f_i. Note the f_i's and T_i^*'s are disjoint parts of T^*, but the subpaths may overlap inside P_1.

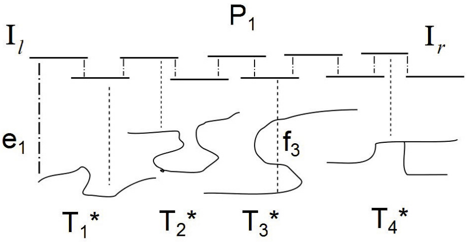
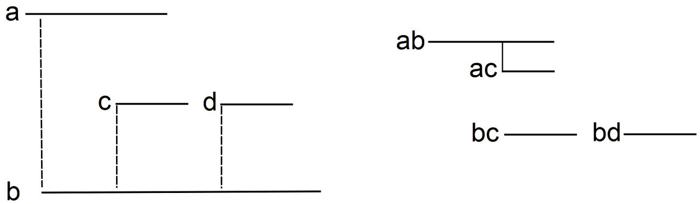


Fig. 2. P_1 and the ${T_i}^*$ subtrees. Each f_i in T^* is replaced by an e_i in T.

An edge $e \in P_1$ appears in at most $k - 1$ of the $P_{1,i}$ subpaths, since each subpath witnesses another vertex (from T_i^*) that must appear in the bag with e. So $\sum_i w(e_i) \leq \sum_i [w(f_i) + w({T_i}^*) + w(P_{1,i})] \leq w(T^*) + (k-1)w(P_1) \leq k \cdot w(T^*)$. Since $w(T) \leq O(k \cdot w(T^*)) + \sum_i w(T_i)$ and $\sum_i w({T_i}^*) \leq w(T^*)$, a simple depth-$k$ recursion finishes our bound. □

Remark: we do not have to compute T as in Lemma 2; it suffices to use any light enough monotone spanning tree. A natural choice is to let T be the *lightest* monotone spanning tree, which we compute as follows. Start with just the root (in the leftmost bag), and grow the tree in a left-to-right scan of the bags: each time a bag B introduces a new vertex v, add an edge connecting v to its nearest neighbor in B (which is already in T).

In the completed G, a *triangle move* is a charging move where a non-tree edge e charges a path P of length two, where at most one edge of P is not in T. We now define $T^{(2)}$, a graph whose edges represent triangle moves. Each vertex jk of $T^{(2)}$ corresponds to an edge $\{j,k\}$ in G. We also represent the vertex jk by the interval $I_{jk} = I_j \cap I_k$. To define the edges of $T^{(2)}$, we first define a *parent* for each vertex jk. If $\{j,k\}$ is an edge of T, then jk has no parent. Otherwise, suppose $\text{left}(j) < \text{left}(k)$ (else swap them), and let i be the parent of k in T; i must exist since k is not the root. Note $\{i,j,k\}$ is a triangle in G. Now we say the parent of jk is ij, and we add the edge $\{ij, jk\}$ in $T^{(2)}$. Note $\text{left}(ij) < \text{left}(jk)$, so these parent links are acyclic. Thus $T^{(2)}$ is a forest, with each component rooted at a vertex corresponding to an edge of T. Figure 3 illustrates a simple monotone tree T and its forest $T^{(2)}$.



(a) a monotone tree T (b) $T^{(2)}$ produced from T

Fig. 3. Horizontal lines are intervals, dashed verticals are edges of T

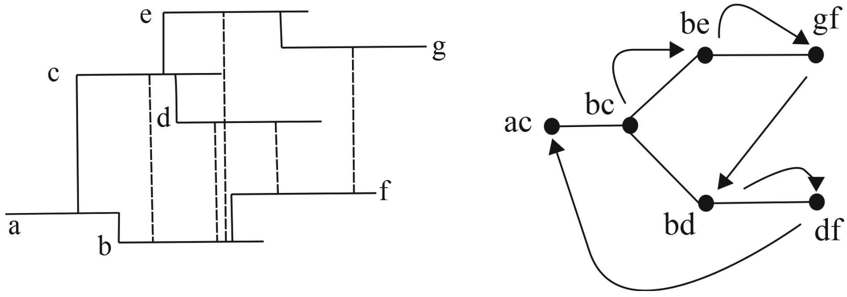
Lemma 3. *Given G as above with a monotone spanning tree T, there is an acyclic charging scheme from G to T of value $O(k)$.*

Proof. Recall $T^{(2)}$ is a forest. Fix a component C of $T^{(2)}$; it is a tree, rooted at a vertex r corresponding to an edge of T, and that is the only such vertex in C. Consider a directed Euler tour of C, traversing each edge twice. Delete each tour edge out of r, so we get a list of directed paths, each of the form

$$e_1 \rightarrow e_2 \rightarrow \cdots e_m \rightarrow r$$

where each vertex e_i corresponds to some edge of $G - T$. Since C is a tree, these paths are vertex disjoint (except at r). However, a vertex may appear more than once on the same path; call an appearance e_i a *repeat* if the same vertex appeared earlier on the path. Let $\mathcal{P}$ be the collection of all these paths, from all components of $T^{(2)}$.

We now propose a charging scheme (which fails to be acyclic). Recall how we constructed edges in $T^{(2)}$: we connect each vertex jk (corresponding to an edge of $G - T$) to its parent ij. If a path in $\mathcal{P}$ traverses this edge in the direction $jk \rightarrow ij$, we add the triangle move where edge $\{j,k\}$ charges one unit to path $j - i - k$. If a path traverses this edge in the other direction $ij \rightarrow jk$ (so ij is



(a) some edges of T (solid) and G-T (dashed)

(b) a shortcut tour in $T^{(2)}$, ending at ac

Fig. 4. Edges of G (left) are vertices of $T^{(2)}$ (right)

not a tree edge), we add the triangle move where edge $\{i, j\}$ charges one unit to path $i - k - j$. In either direction, the tree edge $\{i, k\}$ is charged.

For an edge $e \in G - T$, the corresponding vertex appears at least once on a path, and it has at least as many out-edges as in-edges, so our proposed scheme satisfies conditions (1) and (2). For an edge $e \in T$, we must bound the number of times it is charged. Since G has maxdegree $O(k)$, e appears in $O(k)$ distinct triangles, and it is charged at most twice per triangle (this includes the charges it receives in its role as r). So if we choose $v = O(k)$, condition (3) is satisfied. Also there are no charges out of tree edges, so condition (4) is also satisfied.

However, this charging scheme does not satisfy condition (5); if a vertex (corresponding to an edge $e \in G - T$) has a repeat appearance on its path, then there is no consistent way to order the edges. To fix this, we eliminate all "repeat" appearances using shortcuts. That is, whenever we have a sequence $e_1 \rightarrow e_2 \rightarrow e_3$ where e_2 is a repeat, we shortcut out e_2. Note such shortcuts can be combined. For example if we have a sequence $e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5$, corresponding to four triangle moves, it is possible to shortcut out e_2, e_3, e_4 (in any order), and the result is a single charge from e_1 to a path containing e_5 (the rest of the charged path is all tree edges). After eliminating all repeats by shortcuts, we get the desired acyclic scheme. □

4 Conclusion and Further Work

Regarding our main result, it is not clear whether we really need to force the edges of a monotone T in the greedy spanner computation. Also, we might hope to reduce the $O(k^3)$ factor to something smaller.

The next obvious target is bounded treewidth graphs, a prerequisite for handling clique sums as in the Robertson-Seymour characterization.

There are several obvious directions to try extending the current approach to further minor-closed graph families. First, as extensions of Theorem 1, we propose two open problems: show light spanners for a planar graph with a single vortex, and show light spanners for a path-like clique-sum of planar graphs. For these cases it may help to compose multiple charging schemes into an "ϵ-scheme", as was necessary for apex graphs [8]. As usual, the main difficulty is that we have no control over the MST topology; if the MST has a nice topology (e.g. some form of monotonicity), then we would be done.

Given a bounded treewidth graph, we can still define the notion of a monotone spanning tree T. We choose roots in the decomposition tree and T; whenever vertex v has parent p in T, we require p to be in the bag containing v which is closest to the root. If we can find such a T that is light enough, then we could repeat the rest of our argument from the bounded pathwidth case. However, there is an obstacle: the light monotone tree might not exist.

Theorem 3. *There is an edge-weighted graph G with a bounded treewidth decomposition, such that any monotone spanning tree T in the completion of G has weight $\Omega(\lg n) \cdot MST(G)$.*

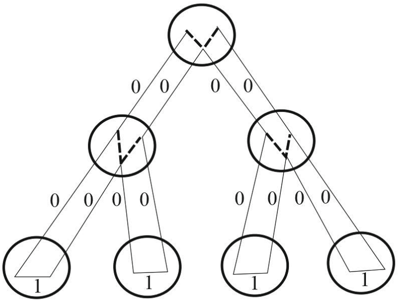


Fig. 5. A bounded treewidth graph with no light monotone tree. The solid path P is the minimum spanning tree. The horizontal edge in each leaf bag has weight one, all other solid edges of P have weight zero. All other edges (in particular, the dashed ones) have weight equal to the distance in P between its endpoints.

Proof. We construct G as follows (see Figure 5). We start with a balanced binary tree with n nodes, think of this as our tree of bags. We assign 3 nodes to each internal bag, and 2 to each leaf bag. We connect these vertices by a path P as shown; each edge of P in a leaf bag has weight one, all other edges in P have weight zero. (Note we must grow our bags a bit to support all these edges of P.)

Now in any monotone tree T for G, for each internal bag, the "bottom" vertex of the three must be connected to one of the other two (its parent in T); in other words, we must pick one of the dashed edges shown in each internal bag.

The main observation is that if we sum up the weights of these selected edges over one level of the decomposition tree, their total is already a constant fraction of $w(P)$. Summing over all levels, the total weight $w(T)$ is $\Omega(\log n) \cdot w(P)$, as claimed. □

References

1. Althöfer, I., Das, G., Dobkin, D., Joseph, D., Soares, J.: On sparse spanners of weighted graphs. Discrete Comput. Geom. 9, 81–100 (1993)
2. Arora, S., Grigni, M., Karger, D.R., Klein, P.N., Woloszyn, A.: A polynomial-time approximation scheme for weighted planar graph TSP. In: SODA, pp. 33–41 (1998)
3. Berger, A., Czumaj, A., Grigni, M., Zhao, H.: Approximation Schemes for Minimum 2-Connected Spanning Subgraphs in Weighted Planar Graphs. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 472–483. Springer, Heidelberg (2005)
4. Berger, A., Grigni, M.: Minimum Weight 2-Edge-Connected Spanning Subgraphs in Planar Graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 90–101. Springer, Heidelberg (2007)
5. Demaine, E.D., Hajiaghayi, M., Kawarabayashi, K.-i.: Contraction decomposition in H-minor-free graphs and algorithmic applications. In: Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC 2011), pp. 441–450, June 6–8 (2011)
6. Demaine, E.D., Hajiaghayi, M., Mohar, B.: Approximation algorithms via contraction decomposition. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, Philadelphia, PA, USA, pp. 278–287. Society for Industrial and Applied Mathematics (2007)
7. Grigni, M.: Approximate TSP in Graphs with Forbidden Minors. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 869–877. Springer, Heidelberg (2000)
8. Grigni, M., Sissokho, P.: Light spanners and approximate TSP in weighted graphs with forbidden minors. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002, Philadelphia, PA, USA, pp. 852–857. Society for Industrial and Applied Mathematics (2002)
9. Klein, P.N.: A linear-time approximation scheme for tsp for planar weighted graphs. In: Proceedings, 46th IEEE Symposium on Foundations of Computer Science, pp. 146–155 (2005)
10. Kloks, T.: Treewidth, Computations and Approximations. LNCS, vol. 842. Springer (1994)
11. Papadimitriouand, C.H., Yannakakis, M.: The Traveling Salesman Problem with distances one and two. Mathematics of Operations Research 18, 1–11 (1993)
12. Robertson, N., Seymour, P.D.: Graph minors. XVI. Excluding a non-planar graph. J. Comb. Theory, Ser. B 89(1), 43–76 (2003)

Planarizing Gadgets for Perfect Matching Do Not Exist*

Rohit Gurjar[1,**], Arpita Korwar[1], Jochen Messner[2], Simon Straub[3], and Thomas Thierauf[2]

[1] IIT Kanpur, India
[2] Aalen University, Germany
[3] University of Ulm, Germany

Abstract. To reduce a graph problem to its planar version, a standard technique is to replace crossings in a drawing of the input graph by planarizing gadgets. We show unconditionally that such a reduction is not possible for the perfect matching problem and also extend this to some other problems related to perfect matching. We further show that there is no planarizing gadget for the Hamiltonian cycle problem.

1 Introduction

The *perfect matching problem* is a very fundamental computational problem (see, e.g., [16, 22]). Edmonds [8] developed a polynomial-time algorithm, but still it is unknown whether there is an efficient *parallel* algorithm for the perfect matching problem, i.e., whether it is in NC. In their seminal result, Mulmuley, Vazirani, and Vazirani [26] isolated a perfect matching by assigning random weights to the edges. This yielded a randomized parallel algorithm for the problem, it is in RNC. A derandomization of this algorithm is a challenging open problem.

There are NC algorithms for the perfect matching problem for special graph classes, for example for regular bipartite graphs [21], dense graphs [3], and strongly chordal graphs [4].

Here we consider *planar graphs*. Planarity is an interesting property with respect to the perfect matching problems, and seems to change the complexity of the problem drastically:

- Valiant [28] showed that counting the number of perfect matchings in a graph is a hard problem, namely it is #P-complete,
- whereas for planar graphs, Kasteleyn [17] showed that a Pfaffian orientation can be computed in polynomial time, which leads to a polynomial time algorithm for counting the number of perfect matchings. Vazirani [29] showed that the problem is in fact in NC.

* Work supported in part by the Indo-German DST-DFG program, DFG grant TH 472/4-1 and DST grant DST/CS/20100251.

** Partially supported by TCS Research Fellowship.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 478–490, 2012.

In contrast, no NC algorithm is known for the *construction* of a perfect matching in planar graphs. This is a puzzling state of affairs because, intuitively, counting seems to be a harder problem than construction. There is, however, an RNC algorithm for the construction problem [26].

Much work has been done on the perfect matching for *bipartite planar* graphs [25, 23, 20, 6, 13, 7]. The current best bound on the problem is unambiguous logspace, UL, for decision and construction [7]. Note that for the bipartite perfect matching problem no better bounds are known than for the general perfect matching problem.

In this paper, we investigate the question of whether there is a logspace or NC reduction from the perfect matching problem to the planar perfect matching problem. It is quite possible that such a reduction exists.

– Such a reduction would be a breakthrough result because it would derandomize the RNC algorithm for perfect matching. Many researchers conjecture that such a derandomization is possible (see, e.g., [1]). Hence, this could be one way of doing it.
– A reduction does not necessarily maintain the number of perfect matchings. Hence, it does not imply an unexpected collapse of complexity classes.

An obvious approach to such a reduction is a *planarizing gadget*: a planar graph that locally replaces the crossing edges of a given drawing of a graph. It is natural to suspect that any more globally acting reduction would be very involved to construct. Examples for planarizing gadgets are the reductions of 3-colorability and vertex cover to their planar versions [10]. In contrast, because of the four color theorem, a planarizing gadget for k-colorability cannot exist for $k \geq 4$. Datta et al. [5] have recently used a planarizing gadget to investigate the complexity of computing the determinant of a matrix, which is the adjacency matrix of a planar graph. They construct a gadget that reduces the general determinant to the planar determinant. Therefore, both problems have the same complexity, they are GapL-complete. The analogous result has been shown for the permanent, again via some planarizing gadget. Therefore, the permanent and the planar permanent are #P-complete.

Our first result is to construct an obstacle in getting an NC algorithm for the perfect matching problem: we show that planarizing gadgets for perfect matching do not exist. We extend the result to unique perfect matching, weighted perfect matching, exact perfect matching, and counting modulo k perfect matching.

The planar Hamiltonian cycle problem was shown to be NP-complete by a direct reduction from 3-SAT [11]. In the Computational Complexity Blog, Gasarch [12] asks whether there is a reduction from HAM to its planar version via some planarizing gadget. In a comment to the blog, David Johnson finds this to be an interesting open problem. Using similar arguments as we used for the perfect matching problem, we give a negative answer to Gasarch's question: there is no planarizing gadget for the Hamiltonian cycle problem. Recently we discovered that this observation was made independently and earlier in a post by Burke [2].

2 Preliminaries

Let $G = (V, E)$ be an undirected graph. A *matching* in G is a set $M \subseteq E$, such that no two edges in M have a vertex in common. A matching M is called *perfect* if every vertex occurs as an endpoint of some edge in M. In the decision problem of *perfect matching*, one has to decide whether G has a perfect matching,

$$PM = \{ G \mid G \text{ has a perfect matching} \}.$$

For a weight function $w : E \rightarrow \mathbb{N}$ of the graph, the *weight of a matching M* is defined as $w(M) = \sum_{e \in M} w(e)$.

Sequential algorithms to compute maximum matchings use *augmenting path techniques* [15]. They are described in many textbooks, see for example [19, 16]. We mention some simple facts. Let M and M' be matchings in a graph $G = (V, E)$. Consider the subgraph $G' = (V, M \triangle M')$ of G that contains only the edges in the symmetric difference of M and M'. This graph consists of *alternating paths (with respect to M and M')*. That is, the paths have edges alternating from M and M'. Note that some of these paths can be cycles (i.e., start and end vertex being the same). Also, they are simple and pairwise disjoint. If M and M' are *perfect matchings* in G, then $M \triangle M'$ consists of alternating cycles only.

Problems. Let us now define the other matching problems which we consider.

- *Unique perfect matching*: Given a graph G, decide whether G has precisely one perfect matching.
- *Weighted perfect matching*: Given a graph G, a weight function w on the edges and a number W, decide whether there is a perfect matching in G of weight at most W.
- *Exact perfect matching*: Given a graph G where every edge is colored either red or blue, and a number k, decide whether there is a perfect matching in G with exactly k red edges.
- *Weighted exact perfect matching*: Given a graph G, a weight function w on the edges, and a number W, decide whether there is a perfect matching in G of weight exactly W.
- Mod_k *perfect matching*: Given a graph G, decide whether the number of perfect matchings in G is not zero modulo k.

The unique perfect matching problem is in P [9]. For bipartite graphs it is in NC [18, 14], and for planar graphs it is also in NC [29]. It is an open problem whether the unique perfect matching problem is in NC.

The weighted perfect matching problem is in P [24, 30]. If the weights are polynomially bounded, then the problem is in NC for planar graphs [29].

The exact perfect matching problem is a very puzzling problem: it is not even known to be in P (see, e.g., [27, 31]). It is known to be in RNC [26] and in NC for planar graphs [29].

The weighted exact perfect matching problem with polynomially bounded weights is (logspace) equivalent to the exact perfect matching problem. To reduce from the latter to the former we do the following: in a given red-blue graph G, assign weight 1 to each red edge and weight 0 to each blue edge. Then a perfect matching with weight k is a perfect matching with k red edges in G. The reduction in the other direction is also simple: in a given weighted graph G, replace each edge $e = (a, b)$ with a simple path of length $2w(e) - 1$ from a to b. Color the edges of the path with red and blue colors alternatingly, such that there are $w(e)$ red and $w(e) - 1$ blue edges. Only polynomial number of edges are added. A perfect matching with W red edges corresponds to a perfect matching of weight W in G.

In contrast, the weighted exact perfect matching problem in general, i.e., with weights exponential in the number of nodes, is NP-complete. This is mentioned in [27] without a proof. In fact, we can present a reduction from the subset sum problem which shows that the problem becomes hard already with a simple underlying graph structure: the problem is NP-complete for weighted graphs that consist of disjoint copies of 4-cycles. Hence, in this case the general problem reduces to the planar problem. As we show, such a reduction is not possible using planarizing gadgets.

The counting class #P is defined as the class of functions that can be written as $acc_M(x) : \Sigma^* \to \mathbb{N}$, where M is a nondeterministic polynomial time Turing machine and $acc_M(x)$ is the number of accepting computations of M on input x. As shown in [28], it is complete for #P to compute $pm(G)$, the number of perfect matchings of a given bipartite graph G [28]. Counting modulo some integer k leads to the complexity class Mod_kP of all problems that can be written as

$$\{\, x \in \Sigma^* \mid acc_M(x) \not\equiv 0 \pmod{k} \}.$$

$\oplus$P is a more common name for Mod_2P. Over GF(2), the permanent of a matrix is the same as the determinant. That is, Mod_2 perfect matching in bipartite graphs can be computed in NC. Therefore, Mod_2 perfect matching is unlikely to be complete for $\oplus$P. On the other hand, it can be seen that Mod_k perfect matching is complete for Mod_kP for every odd $k \geq 3$ (cf. Valiant [28]).

Planarizing Gadgets. Let G be a given non-planar graph and consider a drawing of G in the plane. A *planarizing gadget* is a planar graph which is used to replace crossing edges of this drawing of G as shown in Fig. 1. The gadget graph has four designated vertices $v_1, \ldots, v_4$, called *external vertices* which are identified with the corresponding vertices from the crossing. The other vertices of the gadget are called *internal*.

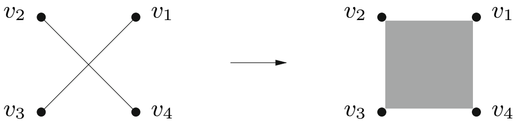


Fig. 1. Planarizing gadget: the two crossing edges on the left are replaced by a planar graph which is indicated by the gray box on the right

The gadget is independent of the structure of the graph. Hence, every crossing of edges is replaced by a copy of the same gadget. Let G' be the resulting planar graph. The gadget is called *planarizing* for a language L of graphs if

$$G \in L \iff G' \in L. \tag{1}$$

More generally L may be a language of pairs $\langle G, k\rangle$, where G is a (possibly weighted) graph and k is a parameter. Then in the planarizing reduction it is suitable to allow a modification of the parameter k with respect to the number of gadgets introduced by the reduction. We call the (possibly weighted) gadget graph *planarizing* for L if $\langle G, k\rangle \in L \iff \langle G', k'\rangle \in L$, where k' may depend on k, the number t of crossings in the considered drawing of G and the number n of nodes. Also, in case of weighted graphs the weights in the gadget may depend on the weights of the crossing edges and the reduction may modify the weights of G using a linear function depending on t and n.

For our purpose where we want to show that no planarizing gadget exists, it suffices to consider the case when each edge crosses at most one other edge. We will show that even for this case there is no planarizing gadget for various languages L.

3 Perfect Matching Problems

First, we look more closely at the properties of a planarizing gadget for perfect matching problems.

Note that it suffices to consider the case where the gadget contains a single edge connected to v_i, for $i \in [4] = \{1, 2, 3, 4\}$. For if there would be several connections from nodes of the gadget to v_i, we could introduce a new node y_i to the gadget and redirect these edges to y_i instead of v_i. Then we add one more node x_i to the gadget and connect it via the path (v_i, x_i, y_i). Now this modified gadget has the structure from Fig. 2 and there is a direct correspondence between the perfect matchings in both gadgets.

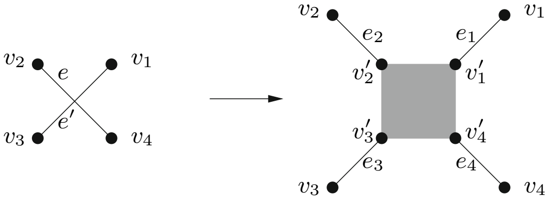


Fig. 2. More details on the planarizing gadget

As shown in Fig. 2, let $e = (v_2, v_4)$ and $e' = (v_1, v_3)$ be the crossing edges in G and let v_i' be the node in the gadget that is connected with v_i via edge e_i, for $i \in [4]$.

Definition 1. *For $I \subseteq [4]$ let $\mathcal{M}_I$ be the set of matchings M of a gadget that cover all internal nodes of the gadget, and $M \cap \{e_1, e_2, e_3, e_4\} = \{\, e_i \mid i \in I\}$.*

The legal matchings *are the matchings that belong to a set in $\mathcal{L}$, where*

$$\mathcal{L} = \{\mathcal{M}_\emptyset, \mathcal{M}_{\{1,3\}}, \mathcal{M}_{\{2,4\}}, \mathcal{M}_{[4]}\}.$$

The illegal matchings *are the matchings that belong to a set in $\mathcal{I}$, where*

$$\mathcal{I} = \{\mathcal{M}_{\{1,2\}}, \mathcal{M}_{\{2,3\}}, \mathcal{M}_{\{3,4\}}, \mathcal{M}_{\{1,4\}}\}.$$

The next Lemma 1 states that in a planarizing gadget for PM legal matchings need to exists and illegal matchings cannot exist which is the actual reason behind naming these classes as legal and illegal. The existence of legal matchings also implies that the gadget needs to have an even number of nodes. This directly implies that $\mathcal{M}_I = \emptyset$ for odd $|I|$ and we do not need to consider these sets.

Lemma 1. *A gadget is planarizing for PM only if*

- *each set in $\mathcal{L}$ is non-empty, and*
- *there is no illegal matching.*

Proof. Consider Fig. 3. Parts (a), (b), and (c) show that $\mathcal{M}_{[4]}$, $\mathcal{M}_{\{2,4\}}$, and $\mathcal{M}_\emptyset$ should be non-empty (respectively). The case $\mathcal{M}_{\{1,3\}}$ is symmetric to $\mathcal{M}_{\{2,4\}}$.

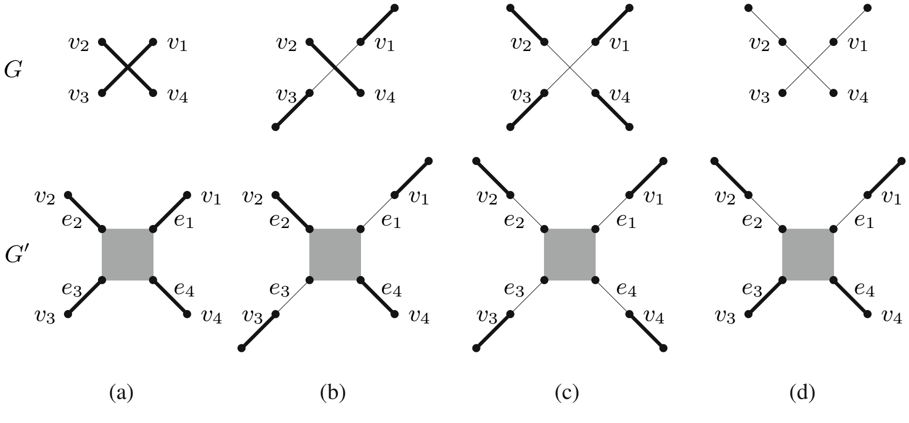


Fig. 3. Graphs G with a perfect matching that contains (a) both, (b) one, and (c) none of the crossing edges. Matching edges are drawn with bold lines. Note that, for graphs G' to have a perfect matching, the gadget should have the legal matchings which contain (a) all four, (b) two opposite, and (c) none of the edges $e_1, \ldots, e_4$. In (d), graph G has no perfect matching. If the gadget would allow the illegal perfect matching that contains e_3, e_4 and not e_1, e_2, then the resulting graph G' would have a perfect matching. Hence, such a gadget does not work.

Part (d) shows that a gadget which allows an illegal matching (a matching in $\mathcal{M}_{\{3,4\}}$) is not planarizing for *PM*. The cases where two other neighboring edges of $e_1, \ldots, e_4$ are used, are symmetric. Therefore, no illegal matching is allowed to exist. □

In the proof of Lemma 1, we argued with the graphs shown in Fig. 3. For simplicity, these graphs are planar, but are drawn with two edges crossing. Clearly, the gadget has to work also in such cases, and hence, we do not need to deal with more complicated non-planar graphs. However, it is easy to extend our graphs to non-planar graphs in such a way, that the perfect matchings are preserved: Let G be one of the above graphs. For every pair of non-adjacent nodes u, v in G, we add two additional nodes $x_{u,v}, y_{u,v}$ which are connected by an edge, and connect u and v with $y_{u,v}$. Let G^* be the resulting graph. Since the only neighbor of $x_{u,v}$ is $y_{u,v}$, every perfect matching in G^* has to use edge $(x_{u,v}, y_{u,v})$. The other edges in the perfect matching are all from G. Hence, perfect matchings in G and G^* differ only by the newly introduced edges $(x_{u,v}, y_{u,v})$.

If G has n nodes, then G^* has the complete graph K_n as minor. Therefore, G^* is non-planar for $n \geq 5$. Only the graph in Fig. 3 (a) has just 4 nodes. But it is easy to enlarge it by a few extra nodes and still cover the same case. Hence, things do not change if we restrict our arguments to non-planar graphs only.

3.1 Perfect Matching

Next, we show that no planarizing gadget for the perfect matching problem exists. The proof constructs an illegal perfect matching out of legal ones.

Theorem 1. *There is no planarizing gadget for the perfect matching problem.*

Proof. Suppose there is a planarizing gadget. We refer to the denotation in Fig. 2 and Definition 1. According to Lemma 1 there are legal matchings $M_{1,3} \in \mathcal{M}_{\{1,3\}}$ and $M_{2,4} \in \mathcal{M}_{\{2,4\}}$.

Consider the subgraph with edges $M_{1,3} \triangle M_{2,4}$ of the gadget: as explained in the preliminary section, $M_{1,3} \triangle M_{2,4}$ consists of some alternating cycles and paths. The nodes v_1, v_2, v_3, v_4 must lie on alternating paths. Since the two matchings cover all nodes in the gadget, there are precisely two disjoint alternating paths p and q, each of which connects two nodes in $\{v_1, v_2, v_3, v_4\}$. The remaining edges of $M_{1,3} \triangle M_{2,4}$ form alternating cycles.

Let p denote the path that contains node v_1. We distinguish three cases:

(i) Suppose that p connects v_1 with v_3. Therefore, q connects v_2 with v_4. As we assume that there is a planar drawing of the gadget where v_1, v_2, v_3, v_4 are placed like in Fig. 2, the two paths must cross in at least one common vertex. Since p and q are disjoint, this is not possible.

(ii) Suppose that $p = p_{1,2}$ connects v_1 with v_2, and $q = p_{3,4}$ connects v_3 with v_4. From $M_{1,3}$ and $M_{2,4}$ we now construct two illegal matchings $M_{2,3}$ and $M_{1,4}$

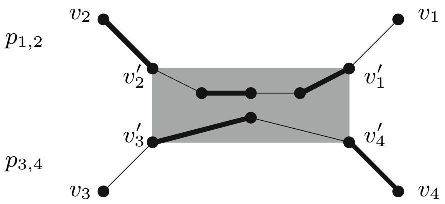


Fig. 4. Matchings $M_{1,3}$ and $M_{2,4}$ are indicated, $M_{2,4}$ with bold edges. The upper alternating path $p = p_{1,2}$ connects v_1 with v_2, the lower path $q = p_{3,4}$ connects v_3 with v_4. The illegal matching $M_{2,3}$ is defined as the bold edges on $p_{1,2}$ and the non-bold edges on $p_{3,4}$ and the other edges from $M_{1,3}$ that are not on these paths. $M_{1,4}$ consists of the remaining edges on both paths and the other edges from $M_{2,4}$.

by exchanging the edges on path $p_{1,2}$ between these two sets. Let $E(p_{1,2})$ denote the set of edges on path $p_{1,2}$. We define

$$M_{2,3} = M_{1,3} \triangle E(p_{1,2}).$$

Similarly we define $M_{1,4} = M_{2,4} \triangle E(p_{1,2})$. Fig. 4 gives an example of the construction. Now both matchings $M_{1,3}$ and $M_{2,4}$ cover each internal node of the gadget, and

- $e_2, e_3 \in M_{2,3}$ and $e_1, e_4 \notin M_{2,3}$ and
- $e_1, e_4 \in M_{1,4}$ and $e_2, e_3 \notin M_{1,4}$.

Hence, $M_{2,3}$ and $M_{1,4}$ are illegal. Therefore, this case is not possible either.

(iii) The case that p connects v_1 with v_4 is analogous to case (ii).

Hence, all cases lead to a contradiction. Therefore, no such gadget exists. □

3.2 Unique Perfect Matching

A planarizing gadget for the unique perfect matching problem needs to have the property that in each of the four legal cases, the matching inside the gadget must be unique, i.e., each set in $\mathcal{L}$ of Definition 1 contains exactly one element. Otherwise, it would not maintain uniqueness in Fig. 3 (a)–(c). However, as shown in the proof of Theorem 1, we cannot avoid getting additional illegal matchings in the gadget. This can be used to destroy the uniqueness in G'. The details can be found in the full version of the paper.

Corollary 1. *There is no planarizing gadget for the unique perfect matching problem.*

3.3 Weighted Perfect Matching

A planarizing gadget for weighted perfect matching may have illegal matchings. But it can be seen that in each class $\mathcal{M} \in \mathcal{L}$ of legal matchings there is a matching

$M \in \mathcal{M}$ whose weight is smaller than the weight of each illegal matching. The proof of Theorem 1 can be extended to show that this is not possible. The details can be found in the full version of the paper.

Corollary 2. *There is no planarizing gadget for the weighted perfect matching problem.*

3.4 Exact Perfect Matching

Corollary 2 says that no planarizing gadget can preserve the minimum weight perfect matching. But it might still be possible that a gadget can preserve some exact weight, which is neither minimum nor maximum.

When replacing crossings of equal weight edges, it can be seen that there are matchings $M_{1,3} \in \mathcal{M}_{\{1,3\}}$ and $M_{2,4} \in \mathcal{M}_{\{2,4\}}$ with some fixed weights and all illegal matchings in the gadget have different weights. The proof of Theorem 1 can be extended to show that the gadget has two illegal matchings $M_{2,3}$, $M_{1,4}$ such that $w(M_{2,3}) + w(M_{1,4}) = w(M_{1,3}) + w(M_{2,4})$.

If a graph G is drawn with $t \geq 2$ crossings, then we will have t gadgets in G'. When the reduction increases the weight by some W_t then there is a combination with two illegal matchings that gives the same increasing weight. We can use this to show that there is no planarizing gadget that works correctly for all graphs. A detailed proof can be found in the full version of the paper.

Theorem 2. *There is no planarizing gadget for the weighted exact perfect matching problem.*

The proofs of Corollary 2 and Theorem 2 already show the non-existence of a planarizing gadget for the case when all edge weights are equal which corresponds to the perfect matching problem. Hence, we can formulate the following corollary.

Corollary 3. *There is no planarizing gadget that reduces the perfect matching problem to the planar weighted perfect matching problem or the planar weighted exact perfect matching problem.*

Similarly the exact perfect matching problem is a special case of the exact weighted perfect matching problem.

Corollary 4. *There is no planarizing gadget for the exact perfect matching problem. Moreover, there is no planarizing gadget that reduces the exact perfect matching problem to the planar weighted exact perfect matching problem.*

3.5 Mod$_k$ Perfect Matching

In the preliminary section we already mentioned that Mod$_k$ perfect matching (for short, Mod$_k$-*PM*), is complete for Mod$_k$P for odd $k \geq 3$. Hence, there is no planarizing gadget for Mod$_k$-*PM* nor any other NC computable planarizing reduction, unless Mod$_k$P = NC, for odd $k \geq 3$. We prove the non-existence of a planarizing gadget independent of the Mod$_k$P $\neq$ NC assumption, for $k \geq 3$.

For a gadget to reduce a graph G to its planarized version G', we must have $pm(G) \equiv 0 \pmod{k}$ if and only if $pm(G') \equiv 0 \pmod{k}$. From the graphs in Fig. 3 it follows that we must have

- $|\mathcal{M}| \not\equiv 0 \pmod{k}$ for all $\mathcal{M} \in \mathcal{L}$ and
- $|\mathcal{M}| \equiv 0 \pmod{k}$ for all $\mathcal{M} \in \mathcal{I}$.

A planarizing gadget for Mod$_2$-*PM* has been provided by [5]. In the following lemma we observe that the legal types of matching classes all have the same size modulo k, say a, and a is relatively prime to k. The proof is omitted here.

Lemma 2. *For a planarizing gadget for* Mod$_k$*-PM there is a number a such that $|\mathcal{M}| \equiv a \pmod{k}$ for all $\mathcal{M} \in \mathcal{L}$. Moreover,* $\gcd(a,k) = 1$.

Our next goal is to construct a bijection between pairs of legal and illegal matchings of a gadget. Recall the proof of Theorem 1: we started with two legal matchings $M_0 \in \mathcal{M}_{1,3}$ and $M_1 \in \mathcal{M}_{2,4}$. Then we defined p to be the alternating path in $M_0 \triangle M_1$ that contains node v_1, and matchings $M_2 = M_0 \triangle E(p)$ and $M_3 = M_1 \triangle E(p)$. Path p either ends in v_2 or in v_4.

- If p ends in v_2 then $M_2 \in \mathcal{M}_{2,3}$ and $M_3 \in \mathcal{M}_{1,4}$,
- if p ends in v_4 then $M_2 \in \mathcal{M}_{3,4}$ and $M_3 \in \mathcal{M}_{1,2}$.

Now observe that this process is reversible: we have $M_2 \triangle M_3 = M_0 \triangle M_1$. That is, $M_2 \triangle M_3$ defines the same alternating path p through v_1 and $M_2 \triangle E(p) = M_0$ and $M_3 \triangle E(p) = M_1$.

The same argument will work if we start with legal matchings $M_0 \in \mathcal{M}_\emptyset$ and $M_1 \in \mathcal{M}_{[4]}$. Hence, we constructed a bijection between the following sets:

$$\begin{aligned} \mathcal{S} &= (\mathcal{M}_\emptyset \times \mathcal{M}_{[4]}) \;\cup\; (\mathcal{M}_{\{1,3\}} \times \mathcal{M}_{\{2,4\}}) \\ \mathcal{T} &= (\mathcal{M}_{\{1,2\}} \times \mathcal{M}_{\{3,4\}}) \;\cup\; (\mathcal{M}_{\{1,4\}} \times \mathcal{M}_{\{2,3\}}) \end{aligned}$$

We conclude:

Lemma 3. *For a planarizing gadget we have $|\mathcal{S}| = |\mathcal{T}|$.*

Theorem 3. *There is no planarizing gadget for* Mod$_k$*-PM for $k \geq 3$.*

Proof. By Lemma 2, we have $|\mathcal{S}| \equiv 2a^2 \pmod{k}$. Since $\mathcal{T}$ contains only illegal classes of matchings, we have $|\mathcal{T}| \equiv 0 \pmod{k}$. By Lemma 3, it follows that $2a^2 \equiv 0 \pmod{k}$. But since $\gcd(a,k) = 1$, this is not possible for $k \geq 3$. □

4 Hamiltonian Cycle

A *Hamiltonian cycle* in graph G is a simple cycle that visits every node in G. The Hamiltonian cycle problem, HAM, is to decide whether a given graph G has a Hamiltonian cycle. A proof can be found in the full version of the paper.

Theorem 4. *There is no planarizing gadget for the Hamiltonian cycle problem.*

In a straightforward way one can modify the proof of Theorem 4 to obtain similar results for the (directed) Hamiltonian path problem and the directed Hamiltonian cycle problem.

Corollary 5. *There is no planarizing gadget for the directed Hamiltonian cycle problem nor for the (directed) Hamiltonian path problem.*

A similar argument shows that there is no planarizing gadget for reachability.

5 Discussion

Our approach allowed us to show unconditionally that there are no planarizing gadgets for various graph problems. Clearly, this does not imply that there is no logspace reduction from the general problem to its planar version. For example, for the Hamiltonian cycle problem or the exact weighted perfect matching problem with large weights, the general and the planar versions are both NP-complete. Nonetheless, we think that the observations are interesting and give some new insight into the problems. Moreover, for the problems like perfect matching where it is not clear whether the problem reduces to its planar version, we eliminated some plausible approach to a reduction.

In our approach, we assumed that the planarizing gadget should work for basically every drawing of the input graph in the plane. A major improvement would be to show that there is no (logspace computable) drawing of the input graph for which a planarizing gadget exists. In fact such a statement can be made for k-colorability with $k \geq 4$: there is no planarizing gadget for the 5-clique K_5, irrespective of the drawing of K_5. Such a gadget would guaranty that the planarized version K_5' is non-4-colorable, which is not possible. On the other hand, such an unconditional statement does not hold for the perfect matching problem nor for the Hamiltonian cycle problem. For these problems there are drawings that allow a planarizing gadget: If one is able to compute a Hamiltonian cycle while computing a drawing of a graph, one can draw the graph such that all edges that belong to the Hamiltonian cycle do not cross any other edge (start by drawing the cycle as circle). Similarly, for the perfect matching problem there is a drawing where matching edges do not have crossings. For such drawings the empty graph is a planarizing gadget (just remove the crossing edges). The following question arises: if one assumes that the Hamiltonian cycle, resp. the perfect matching, of a graph G cannot be computed in logspace, can one show that there is no planarizing gadget for any logspace computable drawing of G?

Acknowledgment. We thank an anonymous referee for his helpful comments.

References

[1] Agrawal, M.: On derandomizing tests for certain polynomial identities. In: Proceedings of the Conference on Computational Complexity, pp. 355–359 (2003)
[2] Burke, K.: http://cstheory.stackexchange.com/questions/9587 (2012)

[3] Dahlhaus, E., Hajnal, P., Karpinski, M.: On the parallel complexity of Hamiltonian cycles and matching problem in dense graphs. J. Algorithms 15, 367–384 (1993)

[4] Dahlhaus, E., Karpinski, M.: Matching and multidimensional matching in chordal and strongly chordal graphs. Disc. Appl. Math. 84, 79–91 (1998)

[5] Datta, S., Kulkarni, R., Limaye, N., Mahajan, M.: Planarity, determinants, permanents, and (unique) matchings. ACM Trans. Comput. Theory, 10:1–10:20 (2010)

[6] Datta, S., Kulkarni, R., Roy, S.: Deterministically isolating a perfect matching in bipartite planar graphs. Theor. Comput. Syst. 47, 737–757 (2010)

[7] Datta, S., Kulkarni, R., Tewari, R.: Perfect matching in bipartite planar graphs is in UL. Technical Report TR10-201, ECCC (2011)

[8] Edmonds, J.: Paths, trees, and flowers. Can. J. Math. 17, 449–467 (1965)

[9] Gabow, H.N., Kaplan, H., Tarjan, R.E.: Unique maximum matching algorithms. J. Algorithms 40(2), 159–183 (2001)

[10] Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete graph problems. Theor. Comput. Sci. 1(3), 237–267 (1976)

[11] Garey, M.R., Johnson, D.S., Tarjan, R.E.: The planar Hamiltonian circuit problem is NP-complete. SIAM J. Comput 5(4), 704–714 (1976)

[12] Gasarch, W.: Is there a nice gadget for showing planar HC is NPC? Computational Complexity Blog (2012), http://blog.computationalcomplexity.org/2012/01/is-there-nice-gadget-for-showing-planar.html

[13] Hoang, T.M.: On the matching problem for special graph classes. In: Proceedings of the Conference on Computational Complexity, pp. 139–150 (2010)

[14] Hoang, T.M., Mahajan, M., Thierauf, T.: On the Bipartite Unique Perfect Matching Problem. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 453–464. Springer, Heidelberg (2006)

[15] Hopcroft, J., Karp, R.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. 2, 225–231 (1973)

[16] Karpinski, M., Rytter, W.: Fast Parallel Algorithms for Graph Matching Problems. Oxford University Press (1998)

[17] Kasteleyn, P.W.: Graph theory and crystal physics. In: Harary, F. (ed.) Graph Theory and Theoret. Physics, pp. 43–110. Academic Press (1967)

[18] Kozen, D.C., Vazirani, U.V., Vazirani, V.V.: NC Algorithms for Comparability Graphs, Interval Graphs, and Testing for Unique Perfect Matchings. In: Maheshwari, S.N. (ed.) FSTTCS 1985. LNCS, vol. 206, pp. 496–503. Springer, Heidelberg (1985)

[19] Kozen, D.: The Design and Analysis of Algorithms. Springer (1991)

[20] Kulkarni, R., Mahajan, M., Varadarajan, K.: Some perfect matchings and perfect half-integral matchings in NC. Chic. J. Theor. Comput. 2008(4) (2008)

[21] Lev, G., Pippenger, M., Valiant, L.: A fast parallel algorithm for routing in permutation networks. IEEE Trans. Computers 30, 93–100 (1981)

[22] Lovasz, L., Plummer, M.D.: Matching theory. North-Holland (1986)

[23] Mahajan, M., Varadarajan, K.R.: A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs. In: 32th ACM Symp. Theo. Comput (STOC), pp. 351–357. ACM Press (2000)

[24] Micali, S., Vazirani, V.: An $O(\sqrt{|v|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In: FOCS, pp. 17–27 (1980)

[25] Miller, G.L., Naor, J.S.: Flow in planar graphs with multiple sources and sinks. SIAM J. Comput. 24, 1002–1017 (1995)

[26] Mulmuley, K., Vazirani, U., Vazirani, V.: Matching is as easy as matrix inversion. Combinatorica 7, 105–113 (1987)

[27] Papadimitriou, C.H., Yannakakis, M.: The complexity of restricted spanning tree problems. J. ACM 29, 285–309 (1982)

[28] Valiant, L.: The complexity of computing the permanent. Theoretical Computer Science 8, 189–201 (1979)

[29] Vazirani, V.: NC algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. Inf. Comput. 80, 152–164 (1989)

[30] Vazirani, V.: A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. Combinatorica 14, 71–109 (1994)

[31] Yuster, R.: Almost exact matchings. Algorithmica 63(1-2), 39–50 (2012)

Kernels for Edge Dominating Set: Simpler or Smaller

Torben Hagerup

Institut für Informatik, Universität Augsburg, 86135 Augsburg, Germany
hagerup@informatik.uni-augsburg.de

Abstract. A *kernelization* for a parameterized computational problem is a polynomial-time procedure that transforms every instance of the problem into an equivalent instance (the so-called *kernel*) whose size is bounded by a function of the value of the chosen parameter. We present new kernelizations for the NP-complete Edge Dominating Set problem which asks, given an undirected graph $G = (V, E)$ and an integer k, whether there exists a subset $D \subseteq E$ with $|D| \leq k$ such that every edge in E shares at least one endpoint with some edge in D. The best previous kernelization for Edge Dominating Set, due to Xiao, Kloks and Poon, yields a kernel with at most $2k^2 + 2k$ vertices in linear time. We first describe a very simple linear-time kernelization whose output has at most $4k^2 + 4k$ vertices and is either a trivial "no" instance or a vertex-induced subgraph of the input graph in which every edge dominating set of size $\leq k$ is also an edge dominating set of the input graph. We then show that a refinement of the algorithm of Xiao, Kloks and Poon and a different analysis can lower the bound on the number of vertices in the kernel by a factor of about 4, namely to $\max\{\frac{1}{2}k^2 + \frac{7}{2}k, 6k\}$.

1 Introduction

1.1 Problem Statement

An active research direction in the area of parameterized computation is the search for kernelization algorithms for hard computational problems (see, e.g., [2]). Briefly stated, a kernelization algorithm for a parameterized computational problem is a polynomial-time procedure that transforms every instance of the problem into an equivalent instance whose size is bounded by a function of the value of the chosen parameter. More formally, a *kernelization* for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is an alphabet and $\mathbb{N} = \{1, 2, \ldots\}$, is an algorithm $\mathcal{A}$ for which there exist a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a function $f : \mathbb{N} \to \mathbb{N}$ such that, applied to an instance $I = (G, k) \in \Sigma^* \times \mathbb{N}$, $\mathcal{A}$ computes, within $p(|I|)$ steps, an instance $I' = (G', k') \in \Sigma^* \times \mathbb{N}$ with $|I'| \leq f(k)$ and $k' \leq k$ such that $I \in L \Leftrightarrow I' \in L$. Here $|I|$ and $|I'|$ denote the number of symbols in the representation of I and I', respectively, according to some suitable encoding scheme. A kernelization can be valuable in the solution of a hard parameterized problem because, used as a preprocessing routine, it allows the input instance

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 491–502, 2012.

to be replaced by a potentially much smaller kernel before the application of a presumably expensive solution algorithm. Because of the typical role of the kernel as the input to an exponential-time computation, reducing the size of the kernel for an important problem, if only by a constant factor, is considered significant progress.

When e and e' are edges in an undirected graph, e *dominates* e' (and vice versa) if e and e' share at least one endpoint; this is the case, in particular, if $e = e'$. An *edge dominating set* in an undirected graph $G = (V, E)$ is a subset D of E with the property that every edge in E is dominated by at least one edge in D. Put differently, an edge dominating set in G is the same as a dominating set in the usual sense in the line graph of G. Let us write $\mathit{Edom}(G)$ for the *edge domination number* of G, i.e., the size of a smallest edge dominating set in G. This paper studies kernelizations for the problem Edge Dominating Set, formally defined as the language

$$\{(G, k) \mid G \text{ is an undirected graph, } k \in \mathbb{N} \text{ and } \mathit{Edom}(G) \le k\} ,$$

which is NP-complete, even if the input graph G is restricted to be bipartite and of maximum degree 3 [8]. As is common, our terminology does not always distinguish rigorously between the pair (G, k) and the graph G.

1.2 Previous Work

Fernau [1] showed the existence of a linear-time kernelization for Edge Dominating Set that yields a kernel with at most $8k^2$ vertices. Prieto [5] described a quadratic-time kernelization for the closely related Minimum Maximal Matching problem and bounded the number of vertices in the kernel by $4k^2 + 8k$; her algorithm is also a kernelization for Edge Dominating Set. The best previous kernelization for Edge Dominating Set, due to Xiao, Kloks and Poon [7], computes a kernel with at most $2k^2 + 2k$ vertices for $k \ge 2$ and $O(k^3)$ edges in linear time.

1.3 Our Contributions

We first describe a kernelization for Edge Dominating Set that yields a kernel with at most $4k^2 + 4k$ vertices and $O(k^3)$ edges. In terms of kernel size, the new algorithm does not compare favorably with the best kernelization cited above. It does, however, have the following advantages: (1) With the exception of Fernau's algorithm, it is simpler than the earlier kernelizations; (2) the kernel is either a trivial "no" instance or a vertex-induced subgraph of the input graph G; (3) every edge dominating set in the kernel of size at most k is also an edge dominating set in G.

Subsequently we show that a kernel with at most $\max\{\frac{1}{2}k^2 + \frac{7}{2}k, 6k\}$ vertices and at most $\frac{8}{27}k^3 + O(k^2)$ edges can be computed in linear time. Our kernelization builds on that of Xiao, Kloks and Poon [7], but refines and extends it and analyzes the kernel size in a different way. We also show that the bound of $\max\{\frac{1}{2}k^2 + \frac{7}{2}k, 6k\}$ vertices is tight for the algorithm.

2 The Protection Algorithm

This section describes and analyzes a very simple kernelization for Edge Dominating Set, the *protection* algorithm. The protection algorithm is parameterized by a positive integer K. Let the input be a pair (G, k), where $G = (V, E)$ is an undirected graph and $k \in \mathbb{N}$. Define a vertex to be *small* if its degree is at most K, and *large* otherwise.

2.1 Description

The protection algorithm consists of the steps described below. *Protecting* a vertex is a suggestive term for setting an initially cleared bit associated with the vertex.

1. Protect every small vertex that has at least one small neighbor.
2. Protecting at most $K + 2$ additional vertices per large vertex, ensure that every large vertex is protected and has at least $K + 1$ protected neighbors.
3. Obtain a graph G' from G by removing every unprotected vertex.
4. If G' is larger than allowed by the analysis (see below), return a trivial "no" instance; otherwise return (G', k).

To execute the algorithm, first classify each vertex as small or large. Then step through the edges, protecting the endpoints of each if they are both small. Next step through the large vertices in an arbitrary order and, for each large vertex u, protect it, count the current number r of its protected neighbors and, if $r < K + 1$, protect $K + 1 - r$ arbitrary additional neighbors of u—since u is large, of course, it has enough neighbors. (Another possibility is to protect $K + 1$ arbitrary neighbors of u, independently of whether they were already protected.) Finally step through the vertices again, removing each vertex that is not protected, and, depending on the size of the resulting graph G', return G' or a trivial "no" instance. The protection algorithm is easily executed in $O(|V| + |E|)$ time.

2.2 Correctness

Assume that Step 4 does not return a trivial "no" instance and let $G' = (V', E')$ be the graph returned. If D is an edge dominating set in G, we can construct a set $D' \subseteq E'$ by replacing each edge $\{u, v\} \in D$ by an edge $\{u', v'\} \in E'$ such that $u' = u$ if $u \in V'$ and $v' = v$ if $v \in V'$. To see that this is possible, let $\{u, v\} \in D$ and assume, e.g., that $u \in V \setminus V'$, i.e., u is a vertex that is removed when stepping from G to G'. Then, in G, u is small and v is large. But then v has at least $K + 1$ incident edges in G', any one of which can serve as $\{u', v'\}$. If an edge that belongs to E' is dominated in G by an edge $\{u, v\} \in D$, it is dominated in G' by the replacement edge $\{u', v'\} \in D'$ identified above, with which $\{u, v\}$ shares every "surviving" endpoint. Therefore D' is an edge dominating set in G' of size at most $|D|$.

For the converse direction, fix $K = 2k$ and assume that D' is an edge dominating set in G' with $|D'| \leq k$. Note that if a vertex in G' is large, it must be incident on an edge in D'—otherwise dominating all of its incident edges would need more than k edges. A simple argument shows D' to be an edge dominating set not only in G', but also in G: If an edge in E has an endpoint u that is large in G, u is also present and large in G' and, as noted above, one of its incident edges belongs to D'. And if an edge in G has no large endpoint, it occurs also in G' and must be dominated by an edge in D'.

2.3 Kernel Size

Let D' be an edge dominating set in G' of size at most k and let S' be the set of endpoints of edges in D'. By definition, every edge in E' has an endpoint in S', so if a vertex in G' does not belong to S', all of its neighbors do. A small vertex in S' has at most K neighbors, and the processing of a large vertex in S' protects at most $K + 2$ vertices that were not already protected. Therefore $|V'| \leq |S'|(K + 2) \leq 2k(K + 2) = 2k(2k + 2) = 4k^2 + 4k$.

The number of edges in G' with exactly one endpoint in S' is bounded by $|S'|(|V'| - |S'|)$, and the number of edges in G' with both endpoints in S' is bounded by $\binom{|S'|}{2} \leq |S'|^2/2$. Therefore $|E'| \leq |S'|(|V'| - |S'|/2)$. The quantity $|S'|(|V'| - |S'|/2)$ is maximized by choosing $|V'|$ as large as possible, i.e., as $2k(K+2)$, and $|S'|$ as close as possible to $|V'|$, i.e., as $2k$. Thus $|E'| \leq 2k(2k(K+2) - k) = 8k^3 + 6k^2$.

2.4 The Bipartite Case

Specialized to input graphs that are bipartite, the Edge Dominating Set problem essentially becomes the Matrix Domination problem: Given a 0-1-matrix X and a positive integer k, is it possible to choose a set D of at most k positions (row-column intersections) in X, each of which contains a 1, so that every position in X that contains a 1 is in the same row or column as a position in D? Weston [6] described a kernelization for Matrix Domination and proved that it runs in $O(n^3)$ time, where n is the largest dimension of the input matrix, and yields a kernel matrix with dimensions of size $O(k \cdot 2^k)$.

Specialized to bipartite input graphs, the protection algorithm is correct if used with $K = k$, since k edges have at most k endpoints that are neighbors of a fixed vertex. The protection algorithm therefore yields a kernel with at most $2k^2 + 4k$ vertices and at most $4k^3 + 6k^2$ edges. Translated to the setting of Matrix Domination, it works in linear time, $O(n^2)$, and computes a kernel matrix with dimensions of size at most $k^2 + 2k$. Up to lower-order terms, this matches the bound of the best previous kernelization, that of Xiao, Kloks and Poon [7], which, applied to bipartite input graphs, performs only marginally better than on general graphs and may yield a kernel with $2k^2 - 2k + 6$ vertices.

3 The Algorithm of Xiao, Kloks and Poon

Recall that a *matching* in an undirected graph G is a set M of edges in G, no two of which share an endpoint. A vertex in G is called *free* with respect to M if it is not an endpoint of any edge in M. A path P in G is *alternating* with respect to M if exactly one of every two consecutive edges on P belongs to M, and a path in G is *augmenting* with respect to M if it is alternating, begins and ends at a free vertex, and is of length at least 1. For $r \geq 1$, let us say that M is *r-maximal* if there is no augmenting path of length $\leq r$ with respect to M. A *maximal* matching is the same as a 1-maximal matching.

The following description of the algorithm of Xiao, Kloks and Poon [7] fixes certain details that the original description left unspecified or seemingly incorrect. The algorithm first computes a maximal matching M_0 in the input graph $G = (V, E)$ and defines m as $|M_0|$, V_m as the set of endpoints of the edges in M_0 and $V^* = V \setminus V_m$ as the set of free vertices. If $m \leq k$, it returns the pair $(G[M_0], k)$, where $G[M_0]$ is the graph induced by the edges in M_0. Otherwise it proceeds to compute A' as the set that consists of each vertex in V_m whose neighbors in V^* number more than $2k - m$ or include a vertex of degree 1. Subsequently it obtains a graph $G' = (V', E')$ from G by removing each vertex in V^* all of whose neighbors are contained in A' and giving each vertex in A' a new neighbor of degree 1. If the size of G' is within the bounds established by the analysis for graphs of edge domination number at most k, G' is returned together with the original parameter value k; otherwise a trivial "no" instance is returned.

Xiao, Kloks and Poon state that if $Edom(G) \leq k$, then $|V'| \leq 2k^2 + 2k$ and $|E'| = O(k^3)$. We mention without proof that their kernelization does not perform much better than indicated: The largest number of vertices that the kernel can have is exactly $\max\{2k^2 - k + 4, 6k\}$ for all $k \in \mathbb{N}$.

4 The 3-MM Algorithm

This section describes and analyzes our best kernelization for Edge Dominating Set, the 3-*MM algorithm*.

4.1 Description

We change the algorithm of Xiao, Kloks and Poon [7] in two ways. The first change, which motivated the name of the new algorithm, is to choose the matching M_0 slightly more carefully. Instead of allowing M_0 to be an arbitrary maximal matching, we require it to be 3-maximal. This change by itself lowers the number of vertices in the kernel to $k^2 + k + 3$ for $k \geq 5$. The second change is embodied in the following *reduction rule*:

(∗) If a vertex $u \in V_m \setminus A'$ belongs to $V_m \setminus A$ for some set $A \subseteq V_m$ such that u has more than $|A|$ neighbors in V^* whose only neighbor in $V_m \setminus A$ is u, then u may be added to A'.

Testing the condition of $(*)$ for all $A \subseteq V_m$ is too expensive. Instead we describe a specific way of applying $(*)$ repeatedly for certain selected sets A (which are, in fact, the successive values taken on by the set A') until this no longer allows A' to be extended. The second change to the algorithm is to replace the original set A' of Xiao, Kloks and Poon by the set obtained in this manner. This change by itself lowers the number of vertices in the kernel to $\lfloor \frac{9}{8}k^2 + \frac{7}{4}k + \frac{25}{8} \rfloor$ for $k \geq 3$.

4.2 Correctness

Our first change to the algorithm, the insistence that M_0 be 3-maximal, does not jeopardize its correctness, as established by Xiao, Kloks and Poon [7], but the second change needs justification.

The correctness proof of the kernelization of [7] hinges on the fact that if the edge domination number of the input graph G is at most k, then the set A' computed by the algorithm is contained in the set of endpoints of some edge dominating set in G of size at most k. The authors prove this and in fact prove that A' is contained in the set of endpoints of *every* edge dominating set in G of size at most k. It can be seen from their proof that the algorithm remains correct even if A' is computed in a different way, provided that A' remains contained in the set of endpoints of some edge dominating set in G of size at most k.

As observed by Harary [3], if $Edom(G) \leq k$, then G contains even an *independent* edge dominating set, i.e., one whose edges form a matching, of size at most k. Let D be such a set and, after zero or more applications of $(*)$, suppose that u is a vertex in $V_m \setminus A'$ and that $A \subseteq V_m$ is a set such that u belongs to $V_m \setminus A$ and has more than $|A|$ neighbors in V^* whose only neighbor in $V_m \setminus A$ is u. Because D is independent, the edges in D incident on vertices in A but not on u dominate at most $|A|$ edges between u and vertices in V^*. Moreover, V^* is independent. Therefore, unless u has an incident edge in D, its incident edges are not all dominated by D. This proves that it is correct to insert u in A'.

4.3 Running Time

To compute a 3-maximal matching M_0 in an undirected graph $G = (V, E)$ in linear time, proceed as follows: In a first phase, obtain a matching M in G by stepping through the edges of E and including each in M exactly if both of its endpoints are free with respect to (the current) M. This is just the usual greedy computation of a maximal matching, and it takes $O(|V|+|E|)$ time if each vertex stores whether it is free. Then, in a second phase, step through the edges in a copy M' of M and process each $\{u, v\} \in M'$ as follows: If G contains two distinct free vertices x and y such that x is a neighbor of u and y is a neighbor of v, choose such vertices x and y and replace the edge $\{u, v\}$ in M by the two edges $\{x, u\}$ and $\{v, y\}$. It is easy to process $\{u, v\}$ in time proportional to the sum of the degrees of u and v, which sums to $O(|E|)$ over the entire execution. Note, in particular, that the requirement $x \neq y$ can always be satisfied if u and v each propose at least two candidates for x and y, respectively, so that no expensive element-uniqueness test is necessary.

One easily observes that M remains a matching throughout the computation. The final value M_0 of M is 3-maximal. To see this, assume otherwise and let P be an augmenting path with respect to M_0 of length at most 3. Note that a vertex, once it stops being free during the computation, never again becomes free. This implies that M' and M_0 are maximal, since two adjacent vertices u and v are not both free after the processing of $\{u, v\}$ in the first phase. Therefore let P contain the vertices x, u, v, y in this order. The edge $\{u, v\} \in M_0$ cannot belong to M', since otherwise its processing in the second phase would have triggered the replacement in M of $\{u, v\}$ by $\{x, u\}$ and $\{v, y\}$ or by two other edges, following which $\{u, v\}$, as an edge without a free endpoint, could not have reentered M. Therefore $\{u, v\}$ was inserted in M during the processing of some edge $e \in M'$ in the second phase. But then either u or v was free immediately before this processing of e. Since x and y are free even at the end of the computation, this contradicts the maximality of M'.

In order to apply the reduction rule $(*)$ repeatedly in linear overall time, store the following information: (1) For each vertex $v \in V^*$, the number $N_1[v]$ of its neighbors in $V_m \setminus A'$; (2) for each vertex $u \in V_m \setminus A'$, the number $N_2[u]$ of those of its neighbors in V^* for which it is the only neighbor in $V_m \setminus A'$; (3) the set T of vertices $u \in V_m \setminus A'$ at which $(*)$ is applicable with A equal to the current or a past value of A'.

The set A' is initialized to the value that it has in the algorithm of Xiao, Kloks and Poon, and (1)–(3) above are initialized accordingly. E.g., for each $u \in V_m \setminus A'$, $N_2[u]$ is initialized to the number of neighbors v of u in V^* with $N_1[v] = 1$, and T is initialized to $\{u \in V_m \setminus A' \mid N_2[u] > |A'|\}$. Then, as long as $T \neq \emptyset$, an arbitrary vertex u is moved from T to A' (i.e., $(*)$ is applied at u), and $N_1[v]$ is decreased by 1 for each neighbor v of u in V^*. For each $v \in V^*$ such that $N_1[v]$ reaches the value 1, $N_2[w]$ is incremented by 1, where w is the single neighbor of v in $V_m \setminus A'$, and if subsequently $N_2[w] > |A'|$, w is inserted in T. All of this can be done in time proportional to the degree of u, and it can be seen to update (1)–(3) correctly. Since every vertex is inserted in A' at most once, the total time needed is $O(|V| + |E|)$. When $T = \emptyset$, $(*)$ is no longer applicable with $A = A'$ at any vertex u.

4.4 The Number of Vertices

The analysis is facilitated by the following elementary observations: If a quadratic polynomial has two real roots s and t, then it assumes its unique extremum at $(s + t)/2$. If a cubic polynomial p has two real roots s and t and s is a double root, then p has local extrema exactly at s and at $(s + 2t)/3$.

Denote by A_0 the final value of the set A'. Assume that $Edom(G) \leq k$, let D be an independent edge dominating set in G with $|D| \leq k$ and let V_D be the set of endpoints of edges in D. Every edge in G has both an endpoint in V_m and an endpoint (possible the same one) in V_D. It follows that every neighbor in G of a vertex $u \in V^* \setminus V_D$ belongs to $V_m \cap V_D$. Since such a vertex u is dropped in the transition from G to G' if its neighborhood is contained in A_0, in G' every vertex in $(V^* \cap V') \setminus V_D$ has a neighbor in the set $B = (V_m \cap V_D) \setminus A_0$. Every edge in M_0 has an endpoint in V_D, which implies that $|V_m \setminus V_D| \leq m$ and that $|V_D \setminus V_m| \leq 2k - m$; in particular, $m \leq 2k$. Since, in G, every neighbor in V^* of a vertex in $V_m \setminus V_D$ belongs to $V_D \setminus V_m$ and has at least one other neighbor (namely in V_D), the relation $|V_D \setminus V_m| \leq 2k - m$ in turn shows that the initial value of A' is contained in $V_m \cap V_D$. A vertex $u \in V_m \setminus V_D$ cannot enter A' in an application of the reduction rule $(*)$ either. For assume otherwise and consider the situation just before u is inserted in T. Let U be the set of neighbors of u in V^* whose only neighbor in $V_m \setminus A'$ is u. By assumption, $|U| > |A'|$. As noted above, $U \subseteq V_D \setminus V_m$, so each vertex in U has an incident edge in D, and the $|U|$ other endpoints of these edges are all distinct. Because they all belong to $V_m \setminus \{u\}$, they must also all belong to A', which is impossible since $|U| > |A'|$. We may conclude that $A_0 \subseteq V_m \cap V_D$. The relationship between some of the sets considered above is illustrated in Fig. 1.

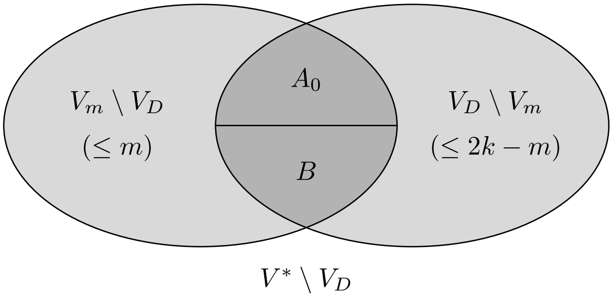


Fig. 1. The main vertex sets of relevance to the analysis. In both G and G', every edge has an endpoint in $A_0 \cup B = V_m \cap V_D$ or goes between $V_m \setminus V_D$ and $V_D \setminus V_m$.

In G', every vertex in A_0 has just one neighbor in $V' \setminus V$. If $m \geq 2k - 1$, by definition of A_0, every vertex in B has at most one neighbor in $V^* \cap V'$. On the other hand, every vertex in G' outside of $V_m \cup V_D$ has a neighbor in $A_0 \cup B$. If $m \geq 2k - 1$, therefore, $|V'| \leq |V_m \cup V_D| + |(V^* \cap V') \setminus V_D| + |V' \setminus V| \leq (|V_m| + |V_D| - |A_0| - |B|) + |B| + |A_0| = |V_m| + |V_D| \leq 2m + 2k \leq 6k$. In the rest of the analysis, assume that $m \leq 2k - 2$.

Lemma 1. *The matching M_0 is 3-maximal not only in G, but also in G'.*

Proof. Since every edge in $E' \setminus E$ is incident on a vertex in V_m, M_0 is maximal in G'. Assume therefore, by way of contradiction, that G' has an augmenting path of length 3 with respect to M_0 that contains the vertices x, u, v, y in this order. Since the only vertices in V_m whose neighborhoods in G and G' differ are those in A_0, u or v must belong to A_0.

In G, u has a neighbor $x' \in V^*$. For every vertex in A_0 has a neighbor in V^*, and if $u \notin A_0$, we can just take $x' = x$. Similarly, in G, v has a neighbor $y' \in V^*$. Now the vertices x', u, v, y', in this order, form an augmenting path in

G, a contradiction, unless $x' = y'$. And if $x' = y'$, there must be an alternative vertex for either x' or y', since otherwise neither u nor v could have entered A'. To see this, observe that none of the three ways of entering A'—having more than $2k - m$ neighbors in V^*, having a neighbor in V^* of degree 1, and having had (for the current or some past value of A') more than $|A'|$ neighbors in V^* with no other neighbor in $V_m \setminus A'$—can apply to a vertex $u \in V_m \setminus A'$ with only one neighbor x' in V^* if x' has another neighbor v in $V_m \setminus A'$; recall, in particular, that $2k - m \geq 2$. □

Write $M_0 = M_1 \cup M_2$, where M_1 is the set of those edges $\{u, v\} \in M_0$ for which, in G', u and v have a common neighbor in $V^* \cap V'$, and $M_2 = M_0 \setminus M_1$. For $i = 1, 2$, let U_i be the set of endpoints of the edges in M_i. By the 3-maximality of M_0 in G', the following three claims are true: (1) In G', every vertex in U_1 has exactly one neighbor in $V^* \cap V'$; (2) $U_1 \cap A_0 = \emptyset$; and (3) every edge in M_2 has at most one endpoint with one or more neighbors in G' outside of V_m. Since, in G', every vertex in A_0 has a neighbor in $V' \setminus V$, this implies that at most $|M_2| - |A_0|$ vertices in $U_2 \setminus A_0$ have one or more neighbors in V^*. Because those that do have at most $2k - m$ neighbors in V^*, the set E^* of edges in G' with one endpoint in $V_m \setminus A_0$ and one endpoint in $V^* \cap V'$ is of size at most

$$|U_1| + (|M_2| - |A_0|)(2k - m) \leq (m - |A_0|)(2k - m) \ .$$

Every vertex in $V^* \cap V'$ is incident on an edge in E^*, and because the reduction rule $(*)$ cannot be applied any further with $A = A_0$, the number of vertices in $V^* \cap V'$ incident on only one edge in E^* (whose other endpoint must belong to $U_2 \setminus A_0$) is bounded by $(m - |A_0|)|A_0|$. Thus

$$|V^* \cap V'| \leq (m - |A_0|) \min\left\{2k - m, |A_0| + \frac{2k - m - |A_0|}{2}\right\}$$

and

$$\begin{aligned} |V'| &\leq |V_m| + |V' \setminus V| + |V^* \cap V'| \\ &\leq 2m + |A_0| + (m - |A_0|) \min\left\{2k - m, \frac{2k - m + |A_0|}{2}\right\} \ . \end{aligned}$$

If $|A_0| \geq 2k - m$ and therefore $m - |A_0| \leq 2(m - k)$,

$$\begin{aligned} |V'| &\leq 3m + (m - |A_0|)(2k - m - 1) \leq 3m + 2(m - k)(2k - m - 1) \\ &= 2m\left(\frac{6k + 1}{2} - m\right) - 4k^2 + 2k \\ &\leq 2\left(\frac{6k + 1}{4}\right)^2 - 4k^2 + 2k = \frac{1}{2}k^2 + \frac{7}{2}k + \frac{1}{8} \ . \end{aligned}$$

On the other hand, if $|A_0| \le 2k-m$ and therefore $2m+|A_0| \le 3k+\frac{1}{2}(m-|A_0|)$,

$$\begin{aligned}|V'| &\le 2m+|A_0|+\frac{1}{2}(m-|A_0|)(2k-m+|A_0|)\\ &\le 3k+\frac{1}{2}(m-|A_0|)(2k+1-(m-|A_0|))\\ &\le 3k+\frac{1}{2}\left(k+\frac{1}{2}\right)^2=\frac{1}{2}k^2+\frac{7}{2}k+\frac{1}{8}\ .\end{aligned}$$

Altogether we have established that $|V'| \le \max\{\lfloor\frac{1}{2}k^2+\frac{7}{2}k+\frac{1}{8}\rfloor, 6k\} = \max\{\frac{1}{2}k^2+\frac{7}{2}k, 6k\}$ for all $k \in \mathbb{N}$. We next show that this bound is tight for the 3-MM algorithm. For arbitrary $k \ge 1$, an input graph that causes the kernel to have $6k$ vertices consists of k disjoint copies of the 'H'-shaped graph shown in Fig. 2(a). It is easy to see that the graph has exactly $6k$ vertices and edge domination number k, and an application of the 3-MM algorithm returns a kernel isomorphic to the input graph.

For even $k \ge 2$, an input graph that may cause the kernel to have $\frac{1}{2}k^2+\frac{7}{2}k$ vertices consists of $k/2$ disjoint copies with $r = k/2$ of the graph shown in Fig. 2(b). The number of vertices is $\frac{k}{2}(2\cdot\frac{k}{2}+7)=\frac{1}{2}k^2+\frac{7}{2}k$, and it is easy to see that the edge domination number is (at most) $\frac{k}{2}\cdot 2 = k$. If the matching M_0 computed by an application of the 3-MM algorithm to the graph consists only of edges drawn horizontally, the kernel returned is isomorphic to the input graph.

For odd $k \ge 3$, finally, consider the disjoint union of $(k-3)/2$ copies of the graph in Fig. 2(b) and a single copy of the graph in Fig. 2(c), all with $r=(k-1)/2$. The number of vertices is

$$\frac{k-3}{2}\left(2\cdot\frac{k-1}{2}+7\right)+\left(4\cdot\frac{k-1}{2}+11\right)=\frac{1}{2}k^2+\frac{7}{2}k\ ,$$

the edge domination number is (at most) $\frac{k-3}{2}\cdot 2+3=k$, and an application of the 3-MM algorithm may produce an isomorphic kernel.

4.5 The Number of Edges

This subsection gives a bound on the number of edges in the kernel that is not tight, but tight for the algorithm with respect to its highest-order term.

When Q and R are disjoint subsets of V', denote by $N(Q)$ the number of edges in G' with both endpoints in Q and by $N(Q,R)$ the number of edges in G' with one endpoint in each of Q and R. Then

$$\begin{aligned}&N(V_m\cup V_D)=\\ &\quad N(V_m\cap V_D)+N(V_m\cap V_D,(V_m\setminus V_D)\cup(V_D\setminus V_m))+N(V_m\setminus V_D,V_D\setminus V_m)\ .\end{aligned}$$

For brevity, put $q=|V_m\cap V_D|$. Then, since $q\ge m$ and $m\le 2k$,

$$\begin{aligned}N(V_m\cup V_D)&\le\binom{q}{2}+q((2m-q)+(2k-q))+(2m-q)(2k-q)\\ &=4km-\frac{q(q+1)}{2}\le\frac{m(8k-1-m)}{2}\le 6k^2-k\ .\end{aligned}$$

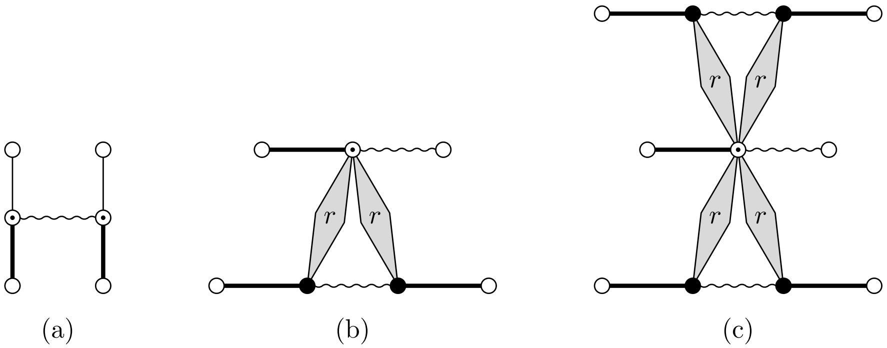


Fig. 2. Subgraphs of worst-case instances for the kernelization. Every gray rhombus represents r degree-2 vertices, each connected to the two vertices at corners of the rhombus. The wavy edges form a minimum edge dominating set D, and the heavy solid edges form a 3-maximal matching M_0. Computed with respect to these, the vertices with a dot and the black vertices are those in A_0 and in B, respectively. The free vertices of degree 1 are removed, but later replaced.

If $m \geq 2k-1$, $N(B, V^* \cap V') \leq |B|$ and $N(A_0, (V^* \cap V') \setminus V_D) \leq |A_0||B|$, so that $|E'| \leq 6k^2 - k + |B| + |A_0||B| + |A_0|$. Since $|A_0| + |B| \leq 2k$, $|B| + |A_0||B| + |A_0| \leq k^2 + 2k$ and $|E'| \leq 7k^2 + k$.

Assume from now on that $m \leq 2k - 2$. According to earlier findings,

$$\begin{aligned} &N(V_m \setminus A_0, V^* \cap V') \leq (m - |A_0|)(2k - m) \quad \text{and} \\ &N(A_0, V^* \cap V') \leq |A_0|(m - |A_0|)(2k - m) \ . \end{aligned}$$

Let $b = m - |A_0|$. Then $|E'|$ is bounded by

$$\begin{aligned} &N(V_m \cup V_D) + N(V_m \setminus A_0, V^* \cap V') + N(A_0, V^* \cap V') + |A_0| \\ \leq\ & \frac{m(8k-1-m)}{2} + b(2k-m) + (m-b)b(2k-m) + m - b \\ \leq\ & \frac{m(8k-1-m+2(2k-m-1)+2)}{2} + (m-b)b(2k-m) \\ \leq\ & 6k^2 + (m-b)b(2k-m) \leq 6k^2 + \frac{1}{4}m^2(2k-m) \ . \end{aligned}$$

The polynomial $m^2(2k-m)$ in m is maximal for $m = \frac{4}{3}k$, so $|E'| \leq \frac{8}{27}k^3 + 6k^2$.

The results of this section can be summarized as follows.

Theorem 1. *Applied to a pair (G, k), where $G = (V, E)$ is an undirected graph and $k \in \mathbb{N}$, the 3-MM algorithm computes, in $O(|V| + |E|)$ time, an undirected graph G' with at most $\max\{\frac{1}{2}k^2 + \frac{7}{2}k, 6k\}$ vertices and at most $\max\{\frac{8}{27}k^3 + 6k^2, 7k^2 + k\}$ edges such that (G', k) belongs to* Edge Dominating Set *if and only if (G, k) does.*

5 Open Problems

A problem that remains open is to discover whether there is a kernelization for Edge Dominating Set whose output has $o(k^2)$ vertices. One may note that the worst-case instances illustrated in Fig. 2 can be reduced to size $O(k)$ with the reduction rule $(*)$, even if not in the restricted way employed by the 3-MM algorithm.

Another question that deserves elucidation is what—if anything—can be achieved for Edge Dominating Set by kernelizations that operate without knowledge of k. Such kernelizations (more precisely, "data reduction rules" with an analogous property) are called "parameter-independent" in [2,4]; a better term might be "parameter-oblivious".

References

1. Fernau, H.: edge dominating set: Efficient Enumeration-Based Exact Algorithms. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 142–153. Springer, Heidelberg (2006)
2. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. SIGACT News 38(1), 31–45 (2007)
3. Harary, F.: Graph Theory. Addison-Wesley, Reading (1969)
4. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
5. Prieto, E.: Systematic Kernelization in FPT Algorithm Design. Ph.D. thesis, The University of Newcastle, Callaghan, N.S.W., Australia (2005)
6. Weston, M.: A fixed-parameter tractable algorithm for matrix domination. Inform. Process. Lett. 90, 267–272 (2004)
7. Xiao, M., Kloks, T., Poon, S.H.: New Parameterized Algorithms for the Edge Dominating Set Problem. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 604–615. Springer, Heidelberg (2011)
8. Yannakakis, M., Gavril, F.: Edge dominating sets in graphs. SIAM J. Appl. Math. 38(3), 364–372 (1980)

Categories of Coalgebraic Games

Furio Honsell[1], Marina Lenisa[1], and Rekha Redamalla[2]

[1] Dipartimento di Matematica e Informatica, Università di Udine, Italy
furio.honsell@comune.udine.it, marina.lenisa@uniud.it
[2] Birla Science Centre Hyderabad, India
rrekhareddy@yahoo.com

Abstract. We consider a general notion of *coalgebraic game*, whereby games are viewed as elements of a *final coalgebra*. This allows for a smooth definition of *game operations* (*e.g.* sum, negation, and linear implication) as *final morphisms*. The notion of coalgebraic game subsumes different notions of games, *e.g.* possibly non-wellfounded Conway games and games arising in Game Semantics à la [AJM00]. We define various categories of coalgebraic games and (total) strategies, where the above operations become functorial, and induce a structure of monoidal closed or $*$-autonomous category. In particular, we define a category of coalgebraic games corresponding to AJM-games and winning strategies, and a generalization to non-wellfounded games of Joyal's category of Conway games. This latter construction provides a categorical characterization of the equivalence by Berlekamp, Conway, Guy on *loopy games*.

Keywords: games, strategies, categories of games and strategies, Conway games, AJM-games.

Introduction

In this paper, we consider a general notion of *coalgebraic game*, whereby games are viewed as elements of a *final coalgebra*. This notion of coalgebraic game is general enough to subsume various notions of games, *e.g.* possibly non-wellfounded Conway games [Con01], and games arising in Game Semantics à la [AJM00]. Coalgebraic methods appear very natural and useful in this context, since they allow to abstract away superficial features of positions in games, and to smoothly define *game operations* as *final morphisms*.

The kind of games that we consider are 2-player games of perfect information, the two players being *Left* (L) and *Right* (R). A game is identified with its *initial* position. At any position, there are *moves* for L and R taking to new positions of the game. Contrary to other approaches in the literature, where games are defined as graphs, we view possibly non-wellfounded games as points of a *final coalgebra* of graphs, *i.e. minimal* graphs w.r.t. bisimilarity. This coalgebraic representation is motivated by the fact that the existence of winning/non-losing strategies is invariant w.r.t. graph bisimilarity. We formalize the notion of *play* as a sequence of pairs move-position, and, on top of it, we define a *strategy* as a function on plays. We focus on *total* strategies for a given player, *i.e.* strategies

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 503–515, 2012.

that must provide an answer, if any, for the player. These differ from *partial strategies*, in which the player can refuse an answer and give up the game. In particular, we introduce and study *winning/non-losing* strategies, which provide winning/non-losing plays when played against any counterstrategy.

In our general coalgebraic framework, we define and discuss various *game operations* arising in the literature, *i.e. sum* and *negation* introduced by Conway [Con01] to analyze games such as Go, or Nim, and *linear logic connectives* of Abramsky et al., see *e.g.* [Abr96, AJM00]. Coalgebraically, such operations can be naturally defined as *final morphisms*, and they uniformly and naturally subsume the corresponding original operations, allowing in particular for a comparison of operations arising in different contexts, such as Conway disjunctive sum and tensor sum on AJM-games. Then, on the basis of these operations, we discuss various categorical constructions, in the spirit of [Joy77], which generalize categories of AJM-games as well as Joyal's original category of Conway games. In particular, we provide a general construction of a symmetric monoidal closed category of possibly non-wellfounded games and (total) strategies, which subsumes Joyal's compact closed category as a *full* subcategory. Interestingly, our category characterizes the equivalence on *loopy games* defined in [BCG82].

Constructions generalizing Joyal's category to non-wellfounded games have been previously considered in [Mel09, MTT09], but in the context of partial strategies; hence they subsume Joyal's category as a subcategory, but *not* as a full subcategory, and the equivalence on games induced by the existence of partial strategies becomes trivial. Solutions to the problem of defining a well-behaved category of non-wellfounded games and total strategies have been presented in [HLR11], for the class of non-wellfounded Conway games where all infinite plays are draws. The solution in the present paper is based on a different and more general construction, and it applies to the class of *mixed games* (infinite plays can be either winning for any of the players or draws). To our knowledge, this is the first category of mixed games subsuming Joyal's construction as a full subcategory and capturing the original loopy equivalence of [BCG82].

The coalgebraic notion of game in this paper generalizes the one introduced in [HL11] for characterizing non-wellfounded Conway games. Coalgebraic methods for modeling games have been used also in [BM96], where the notion of *membership game* has been introduced. This corresponds to a subclass of our coalgebraic games, where at any position L and R have the same moves, and all infinite plays are deemed winning for player II (the player who does not start). However, no operations on games are considered in that setting. In the literature, various notions of bisimilarity equivalences have been considered on games, see *e.g.* [Pau00, Ben02]. But, contrary to our approach, such games are defined as *graphs* of positions, and equivalences on graphs, such as trace equivalences or various bisimilarities are considered. Differently, defining games as the elements of a final coalgebra, we directly work up-to bisimilarity of game graphs.

Summary. In Section 1, we introduce our framework of coalgebraic games and strategies, and we instantiate it to Conway games and AJM-games. In Section 2, we introduce and study general game operations, and in Section 3 we present two parametric categories of games and strategies, subsuming as special instances categories arising in Game Semantics as well as Joyal's category of Conway games. Conclusions and directions for future work appear in Section 4.

1 Coalgebraic Games and Strategies

We consider a general notion of 2-player game of perfect information, where the two players are called *Left* (L) and *Right* (R). A game x is identified with its *initial position*; at any position, there are *moves* for L and R, taking to new *positions* of the game. By abstracting from superficial features of positions, games can be viewed as elements of the final coalgebra for the functor $F_{\mathcal{A}}(X) = \mathcal{P}_{<\kappa}(\mathcal{A} \times X)$, where $\mathcal{A}$ is a parametric set of *atoms* which encode information on moves and positions, *i.e.* move names, and the player who has moved, and $\mathcal{P}_{<\kappa}$ is the set of all subsets of cardinality $< \kappa$. The coalgebra structure captures, for any position, the moves of the players and the corresponding next positions.

We work in the category *Set** of sets belonging to a universe satisfying the Antifoundation Axiom, see [FH83, Acz88]. Of course, we could work in the category *Set* of well-founded sets, but we prefer to use *Set** so as to be able to use identities rather than isomorphisms. Formally, we define:

Definition 1 (Coalgebraic Games). *Let $\mathcal{A}$ be a set of atoms with functions:*

(i) $\mu : \mathcal{A} \to \mathcal{M}$ yielding the name of the move (for a set $\mathcal{M}$ of names),
(ii) $\lambda : \mathcal{A} \to \{L, R\}$ yielding the player who has moved.

Let $F_{\mathcal{A}} : Set^* \to Set^*$ be the functor defined by $F_{\mathcal{A}}(X) = \mathcal{P}_{<\kappa}(\mathcal{A} \times X)$ (with usual definition on morphisms), and let $(\mathcal{G}_{\mathcal{A}}, id)$ be the final $F_{\mathcal{A}}$-coalgebra.
A coalgebraic game *is an element x of the carrier $\mathcal{G}_{\mathcal{A}}$ of the final coalgebra.*

The elements of the final coalgebra $\mathcal{G}_{\mathcal{A}}$ are the *minimal* graphs up-to bisimilarity. In the following, we often refer to *coalgebraic games* simply as *games*. We call *player I* the player who starts the game (who can be L or R in general), and *player II* the other. Once a player has moved on a game x, this brings to a new game/position x'. We define the *plays* on x as the sequences of pairs, move-position, from x; moves in a play are *not* necessarily alternating (this generality will be useful in the sequel, in defining operations on games):

Definition 2 (Plays). *A* play *on a game x_0 is a possibly empty finite or infinite sequence of pairs in $\mathcal{A} \times \mathcal{G}_{\mathcal{A}}$, $s = \langle a_1, x_1 \rangle \ldots$ such that $\forall n \geq 0.\ \langle a_{n+1}, x_{n+1} \rangle \in x_n$. We denote by $Play_x$ the set of plays on x and by $FPlay_x$ the set of finite plays.*

The kind of strategies for a given player on which we focus are those that *always* provide an answer, *if any*, of the player to the moves of the opponent player. In this sense, such strategies are "total", opposite to "partial strategies", where the player can possibly refuse an answer and give up the game. Formally, strategies in our framework are partial functions on finite plays ending with a position where the player is next to move, and yielding (if any) a pair in $\mathcal{A} \times \mathcal{G}_{\mathcal{A}}$, consisting of "a move of the given player together with a next position" on the game x. In what follows, we denote by

– $FPlay_x^{LI}$ ($FPlay_x^{RI}$) the set of possibly empty finite plays on which L (R) acts as player I, and ending with a position where R (L) was last to move, *i.e.* $s = \langle a_1, x_1 \rangle \dots \langle a_n, x_n \rangle$, $\lambda a_1 = L$ and $\lambda a_n = R$ ($\lambda a_1 = R$ and $\lambda a_n = L$).
– $FPlay_x^{LII}$ ($FPlay_x^{RII}$) the set of finite plays on which L (R) acts as player II, and ending with a position where R (L) was last to move, *i.e.* $s = \langle a_1, x_1 \rangle \dots \langle a_n, x_n \rangle$, $\lambda a_1 = R$ ($\lambda a_1 = L$) and $\lambda a_n = R$ ($\lambda a_n = L$).

Formally, we define:

Definition 3 (Strategies). *Let x be a game. A strategy σ for LI (*i.e. *L acting as player I) is a partial function $\sigma : FPlay_x^{LI} \to \mathcal{A} \times \mathcal{G}_{\mathcal{A}}$ such that, for any $s \in FPlay_x^{LI}$,*

– $\sigma(s) = \langle a, x \rangle \implies \lambda a = L \ \wedge \ s\langle a, x \rangle \in FPlay_x$
– $\exists \langle a, x \rangle. \ (s\langle a, x \rangle \in FPlay_x \ \wedge \ \lambda a = L) \implies s \in dom(\sigma)$.

Similarly, one can define strategies for players LII, RI, RII.

We are interested in studying the interactions of a strategy for a given player with the (counter)strategies of the opponent player. When a player plays on a game according to a strategy σ, against an opponent player who follows a (counter)strategy σ', a play arises. Formally, we define:

Definition 4 (Product of Strategies). *Let x be a game.*
(i) Let s be a play on x, and σ a strategy for a player in {LI,LII,RI,RII}. Then s is coherent with *σ if, for any proper prefix s' of s, ending with a position where the player is next to move, $\sigma(s') = \langle a, x \rangle \implies s'\langle a, x \rangle$ is a prefix of s.*
(ii) Given a strategy σ on x and a counterstrategy σ', we define the product *of σ and σ', $\sigma * \sigma'$, as the unique play coherent with both σ and σ'.*

Notice that a play arising from the product of strategies is alternating.

We distinguish between *well-founded games*, *i.e.* well-founded sets as elements of the final coalgebra $\mathcal{G}_{\mathcal{A}}$, and *non-wellfounded games*, *i.e.* non-wellfounded sets in $\mathcal{G}_{\mathcal{A}}$. Clearly, strategies on well-founded games generate only finite plays, while strategies on non-wellfounded games can generate infinite plays.

Strategies for a given player, as we have defined so far, simply provide an answer (if any) of the player to all possible moves of the opponent. Intuitively, a strategy is *winning/non-losing* for a player, if it generates winning/non-losing plays against any possible counterstrategy. We take a finite play to be winning for the player who performs the last move. While infinite plays are taken to be winning for L/R or draws. Formally, we define:

Definition 5 (Winning/non-losing Play). *Let $\nu : Play_x \to \{0, 1, -1\}$ be a payoff function defined on plays of a game x.*
(i) A play s is winning *for player L (R) if $\nu(s) = 1$ ($\nu(s) = -1$).*
(ii) A play s is a draw *if $\nu(s) = 0$.*
(iii) A play s is non-losing *for player L (R) if $\nu(s) \in \{0, 1\}$ ($\nu(s) \in \{0, -1\}$).*

Definition 6 (Winning/non-losing Strategy). *Let $\nu : Play_x \to \{0, 1, -1\}$ be a payoff function on x.*
(i) A strategy σ on x for LI (LII) is winning (non-losing) *for LI (LII) if for any strategy σ' for RII (RI), $\nu(\sigma * \sigma') = 1$ ($\nu(\sigma * \sigma') \in \{0, 1\}$).*
(ii) A strategy σ on x for RI (RII) is winning (non-losing) *if for any strategy σ' for LII (LI), $\nu(\sigma * \sigma') = -1$ ($\nu(\sigma * \sigma') \in \{0, -1\}$).*

We will refer to the whole class of coalgebraic games, where plays can be winning or draws, as *mixed games*; and we will call *fixed games* the subclass of games where all plays are winning for one of the players.

The notion of strategy of Definition 3 is quite general, being defined on plays which carry the information on moves and positions. Often, we are interested in considering special classes of strategies, depending either on moves or on positions (or even only on the last move/position). Here we collect the relevant definitions. For any play s, we denote by $s_{|\mathcal{A}}$ the sequence obtained by erasing all positions, and by $s_{|\mathcal{P}}$ the sequence obtained by erasing all moves from s.

Definition 7. *Let σ be a strategy on a game x.*
(i) σ is pos-independent *if $\forall s, s' \in dom(\sigma)$. $(s_{|\mathcal{A}} = s'_{|\mathcal{A}} \implies \sigma(s) = \sigma(s'))$.*
(ii) σ is move-independent *if $\forall s, s' \in dom(\sigma)$. $(s_{|\mathcal{P}} = s'_{|\mathcal{P}} \implies \sigma(s) = \sigma(s'))$.*

1.1 Conway Games

Conway (wellfounded) games are inductively defined in [Con01] as pairs of sets $x = (X^L, X^R)$, where X^L (X^R) is the set of next positions to which L (R) can move. Such games are purely positional, no move names are considered. In [BCG82], non-wellfounded games are considered, called *loopy* or *mixed* games, but these are defined as graphs of positions, rather than sets, *i.e.* graphs up-to bisimilarity. Here we extend the original set-theoretical definition of [Con01], by representing possibly non-wellfounded Conway games as coalgebraic games for $\mathcal{A}$ the two-element set $\{a^L, a^R\}$, where $\mu a^L = \mu a^R = a$, $\lambda a^L = L$ and $\lambda a^R = R$. These correspond to loopy games taken up-to graph bisimilarity. Our coalgebraic approach is motivated by the fact that the existence of winning/non-losing strategies is preserved under graph bisimilarity of loopy games. Winning/non-losing strategies on Conway games correspond to (move-independent) winning/non-losing strategies of Definition 6.

1.2 Game Semantics

In Game Semantics various notions of games are used, here we focus on the basic games à la [Abr96, AJM00], called *AJM-games*. We define an *AJM-game* as a tuple $G = (M_G, \lambda_G, P_G, W_G)$, where M_G is the set of moves, the function $\lambda_G : M_G \to \{O, P\}$ specifies for each move if it is an O (Opponent) or a P (Player) move; O and P move in strict alternation, O starts the game; the set P_G is a non-empty prefix-closed set of finite alternating sequences of moves starting with an O-move, which represents the set of *legal positions*. These correspond to the finite plays in our setting when positions are omitted. We define P_G^∞ as the set of infinite plays, *i.e.* infinite sequences whose finite prefixes are legal positions. The winning condition for a player on a finite play corresponds to the absence of moves for the other player, while any infinite play is fixed to be winning either for O or for P via the predicate W_G, which holds on an infinite play s ($W_G(s) \downarrow$) iff s is winning for P.

The underlying structure of any such game can be represented in our framework by considering the tree of legal positions (plays). This can be viewed as an element of our final coalgebra $\mathcal{G}_\mathcal{A}$, provided we perform a bisimilarity quotient on nodes (since the tree of plays is not necessarily minimal w.r.t. bisimilarity), thus getting the graph of positions. Formally, we represent such games as follows:

– P is player L and O player R, R starts the game;
– the set $\mathcal{A}$ includes atoms a_m for any $m \in M_G$ s.t. $\mu a_m = m$, $\lambda a_m = \lambda_G m$;
– nodes $\{x_p\}_{p \in P_G}$, $x_p = \{(a_m, x_{p'}) \mid p' = pa \in P_G\}$, are taken up-to bisimilarity;
– the initial position is x_ϵ;
– the payoff function ν is defined on infinite plays by $\nu(s) = \begin{cases} 1 & \text{if } W_G(s) \downarrow \\ -1 & \text{otherwise.} \end{cases}$

Coalgebraic games representing AJM-games are fixed and have a special structure: R starts, at any non-ending position only moves for R or L are available, for any move there is at most one arc labeled by that move, and along any path in the game graph R/L moves strictly alternate. We call *strict games* such subclass of coalgebraic games. They form a subcoalgebra of our final coalgebra.

Winning strategies on AJM-games are defined as suitable subsets of the legal positions, see [Abr96] for more details, and hence they only depends on the sequence of moves (pos-independent strategies in our setting). AJM-games together with winning strategies form a $*$-autonomous category $\mathcal{C}$, see [Abr96]. The precise relationship between $\mathcal{C}$ and the corresponding category of coalgebraic games is formalized in Section 3 via an *equivalence of categories.*

2 Game Operations

In this section, we show how to define various operations on coalgebraic games, including *sum*, *negation*, and *linear implication.* In our framework, game operations can be conveniently defined via final morphisms. These capture the structure of compound games; the extra structure of the payoff function on infinite plays of the compound game is obtained inductively from the payoff of the components.

On mixed games, we define a notion of sum, inspired by Conway *disjunctive sum*; while, on fixed games, we define a notion of sum subsuming the *tensor product* of Game Semantics. The two notions of sum have the same coalgebraic structure, and only differ by the definition of the payoff on infinite plays. This neatedly emerges from the analysis carried out in our coalgebraic framework.

Sum. We start by defining the coalgebraic structure of the sum of two games. On the sum game, at each step, the next player selects any of the component games and makes a legal move in that component, the other component remaining unchanged. The other player can either choose to move in the same component or in a different one. Notice that in this way, even if the play on the sum game agrees with turns of L and R, the subplays in the single components may not agree with turns, in general.

Definition 8 (Sum, coalgebraic structure). *The* sum *of two games is given by the final morphism* $+ : (\mathcal{G}_{\mathcal{A}} \times \mathcal{G}_{\mathcal{A}}, \alpha_+) \longrightarrow (\mathcal{G}_{\mathcal{A}}, id)$*, where the coalgebra morphism* $\alpha_+ : \mathcal{G}_{\mathcal{A}} \times \mathcal{G}_{\mathcal{A}} \longrightarrow F_{\mathcal{A}}(\mathcal{G}_{\mathcal{A}} \times \mathcal{G}_{\mathcal{A}})$ *is defined by:*
$\alpha_+(x, y) = \{\langle a, \langle x', y\rangle\rangle \mid \langle a, x'\rangle \in x\} \cup \{\langle a, \langle x, y'\rangle\rangle \mid \langle a, y'\rangle \in y\}$.
That is: $x + y = \{\langle a, x' + y\rangle \mid \langle a, x'\rangle \in x\} \cup \{\langle a, x + y'\rangle \mid \langle a, y'\rangle \in y\}$.

Two kinds of sum arise from the above coalgebraic definition, by suitably defining the payoff function on infinite plays:

(i) *Mixed sum* $\oplus$. This is defined on mixed games, and it is inspired by Conway disjunctive sum. The payoff of an infinite play will be 1 (-1) if all infinite plays in the components have payoff 1 (-1), it will be 0 otherwise.
(ii) *Fixed sum* $\otimes$. This is defined on fixed games and it generalizes the tensor product of Game Semantics. The payoff of an infinite play is 1 (winning for L) iff all infinite subplays in the components have payoff 1, it will be -1 otherwise. This “asymmetric” definition is motivated by the interpretation of the linear logic tensor connective.

Notice that, on both sums, since plays which agree with turns do not necessarily induce subplays on the components which agree with turns, in order to define the payoff on infinite plays, we need the payoff on *all* plays of the components, also those non conformed to turns. This is the reason for such a liberal definition of plays in Section 1. But, if we restrict ourselves to coalgebraic *strict* games, which correspond to games of Game Semantics, then any play on the sum game which agrees with turns induces subplays with the same property in the components. Moreover, the “switching condition” becomes straightforward.

Negation. The *negation* is a unary game operation, which allows us to build a new game, where the roles of L and R are exchanged. Let us assume that the set of atoms $\mathcal{A}$ is closed under an involution operation, *i.e.*, for any $a \in \mathcal{A}$, let $\overline{a} \in \mathcal{A}$ be such that $\lambda\overline{a} = \overline{\lambda a}$, $\nu\overline{a} = -\nu(a)$, $\mu\overline{a} = \mu a$, where $\overline{L} = R$ and $\overline{L} = R$. The coalgebraic definition of game negation is as follows:

Definition 9 (Negation). *The* negation *of a game is given by the final morphism* $^- : (\mathcal{G}_{\mathcal{A}}, \alpha_-) \longrightarrow (\mathcal{G}_{\mathcal{A}}, id)$*, where the coalgebra morphism* $\alpha_- : \mathcal{G}_{\mathcal{A}} \longrightarrow F_{\mathcal{A}}(\mathcal{G}_{\mathcal{A}})$ *is:* $\alpha_-(x) = \{\langle\overline{a}, x'\rangle \mid \langle a, x'\rangle \in x\}$. *That is:* $\overline{x} = \{\langle\overline{a}, \overline{x'}\rangle \mid \langle a, x'\rangle \in x\}$. *The payoff on infinite plays of* $\overline{x}$ *is taken to be opposite to the payoff on* x.

Clearly, winning/non-losing strategies for a given player on x become winning/non-losing strategies for the opponent player on $\overline{x}$, and $\overline{\overline{x}} = x$, *i.e.* negation is involutive. Notice that both mixed and fixed games are closed under negation. But strict games are not, of course.

Linear Implicaton. Using the two notions of sum, and negation, we can now define two *linear implications*:

Definition 10 (Linear Implications). *We define*
(i) on mixed games: the linear implication $x \to y$ *as the game* $\overline{x \oplus \overline{y}}$*;*
(ii) on fixed games: the linear implication $x \multimap y$ *as the game* $\overline{x \otimes \overline{y}}$*.*

Notice that mixed sum satisfies the equality $\overline{x \oplus y} = \overline{x} \oplus \overline{y}$, hence the linear implication $x \to y$ amounts to $\overline{x} \oplus y$, while the corresponding equality for fixed sum does not hold. More precisely, the coalgebraic structure of the game $x \multimap y$ coincides with the coalgebraic structure of $\overline{x} \otimes y$, but the winning condition on infinite plays is different, namely an infinite play is winning for L on $x \multimap y$ iff the subplay on $\overline{x}$ or that on y is infinite and winning for L, while an infinite play is winning for L on $\overline{x} \otimes y$ iff all infinite subplays on $\overline{x}$ and y are winning for L.

3 Game Categories

The fine analysis of game operations carried out in Section 2 allows us to provide two very general categorical constructions arising from such operations on games. In particular, we provide a category $\mathcal{X}_{\mathcal{A}}$ of *fixed games* and *winning strategies*, parametric w.r.t. the set of atoms $\mathcal{A}$, which is $*$-autonomous, and a symmetric monoidal closed category $\mathcal{Y}_{\mathcal{A}}$ of *mixed games*, also parametric w.r.t. $\mathcal{A}$, obtained by analyzing mixed games via pairs of fixed games. A special instance of $\mathcal{X}_{\mathcal{A}}$ is obtained by instantiating $\mathcal{A}$ as shown in Section 1.2, in order to recover (up-to bisimilarity) AJM-games. A significant result that we obtain, which clarifies the relationships between the original AJM-games and their representation in our framework, is an equivalence between the category of games and winning strategies of [Abr96] and our category of strict coalgebraic games and pos-independent strategies. On the other hand, the category $\mathcal{Y}_{\mathcal{A}}$ of mixed games is related to Conway games. Namely, by suitably instantiating $\mathcal{A}$, we get a category whose objects correspond to the (non-wellfounded) mixed Conway games of [BCG82], a full subcategory of which is Joyal's compact closed category. Remarkably, the equivalence on mixed games induced by the morphisms of our category coincides with the equivalence defined in [BCG82] on loopy games. To our knowledge, this is the first category of mixed games subsuming Joyal's construction as a full subcategory and capturing the original loopy equivalence.

The Category $\mathcal{X}_{\mathcal{A}}$ of Fixed Games. The notions of sum and linear implication on fixed games give rise to a $*$-autonomous category $\mathcal{X}_{\mathcal{A}}$ that generalizes categories of AJM-games and winning strategies.

Definition 11 (The Category $\mathcal{X}_{\mathcal{A}}$)
Objects*:* *fixed games.*
Morphisms*:* $\sigma : x \to y$ *winning strategy for LII on* $x \multimap y$*.*

Identities on $\mathcal{X}_{\mathcal{A}}$ are the *copy-cat strategies*, and closure under composition is obtained via the *swivel-chair strategy. I.e.*, given winning strategies for LII, σ

on $x \multimap y$, τ on $y \multimap z$, the composition strategy, $\tau \circ \sigma$ on $x \multimap z$, is obtained by using the "swivel chair", as follows. Assume R opens on $x \multimap z$, playing either in z or in x, *e.g.* assume R opens in z. Then consider the L move given by the strategy τ on $y \multimap z$: if such L move is in z, then we take this as the L answer in the strategy on $\tau \circ \sigma$; if the L move according to σ is in the y component of $y \multimap z$, then, using the "swivel chair", we can view this move as an R move in the y component of $x \multimap y$. Now L has a next move in $x \multimap y$, according to τ. If this move is in the x component, then we take this as the L answer in $\tau \circ \sigma$; otherwise, if the L move is in y, then we use our swivel chair, viewing this as a move of R in the y component on $y \multimap z$, and so on. Since both σ and τ are winning strategies, by the winning condition on infinite plays on the $\multimap$ game, spelled out at the end of Section 2, we are guaranteed that the dialogue between the y components does not go on forever, and eventually the L move according to σ or τ will be on z or x. This is the L answer to the starting R move in the strategy $\tau \circ \sigma$. Then, we go on in the same way, for any possible next R move. Associativity of composition is also proved by a standard argument.

Fixed sum gives rise to a tensor product on $\mathcal{X}_{\mathcal{A}}$, which determines a structure of a symmetric monoidal closed category; in particular the identity $x \otimes y \multimap z = y \multimap (x \multimap z)$ holds, this latter following from the definition of the $\multimap$ game and from the fact that negation is involutive. Negation is also functorial, and, together with tensor, provides a $*$-autonomous structure on $\mathcal{X}_{\mathcal{A}}$, namely we have in particular the identity $x \otimes y \multimap \overline{z} = x \multimap (\overline{y \otimes z})$. Summarizing:

Theorem 1. *The category $\mathcal{X}_{\mathcal{A}}$ is $*$-autonomous.*

The above construction encompasses categories used in Game Semantics. Namely, let $\mathcal{C}$ be the category of AJM-games and winning strategies of [Abr96], and let us instantiate the parameter $\mathcal{A}$ of $\mathcal{X}_{\mathcal{A}}$ with the set of moves M for such games, getting the category $\mathcal{X}_M$. If we consider the subcategory $\mathcal{SX}_M$ of strict games and pos-independent winning strategies, then we obtain the following equivalence of categories:

Theorem 2. *The category $\mathcal{SX}_M$ of strict games and pos-independent winning strategies is equivalent to the category $\mathcal{C}$ of AJM-games and winning strategies.*

Proof. The equivalence between the categories $\mathcal{SX}_M$ and $\mathcal{C}$ is given by the functor $H : \mathcal{C} \to \mathcal{SX}_M$, which, for a game in $\mathcal{C}$, yields the coalgebraic game obtained by performing a bisimilarity quotient on the tree of legal positions. Winning strategies of $\mathcal{C}$, which are defined on legal positions (*i.e.* plays where positions are omitted, in our setting) are naturally mapped to pos-independent winning strategies in $\mathcal{SX}_M$, providing a one-to-one correspondence. Moreover, each strict coalgebraic game is the image of an AJM-game via H. □

The Category $\mathcal{Y}_{\mathcal{A}}$ of Mixed Games. Defining a category of mixed games and non-losing strategies is not straightforward, the reason being that non-losing strategies are *not* closed under composition. The situation has been analyzed in [HLR11] for hypergames, *i.e.* non-wellfounded Conway games where all infinite plays are draws. The solution proposed there is to restrict the class of morphisms to non-losing *fair* strategies. This category is symmetric monoidal with the mixed sum $\oplus$ as tensor product, but it is *not* monoidal closed; moreover the categorical construction does not immediately extend to the whole class of mixed games. Here, by exploiting the investigation on operations carried out in Section 2, we propose a different solution, which is inspired by the analysis of mixed Conway games x in terms of pairs of fixed games, $\langle x^-, x^+ \rangle$, [BCG82]. The idea is to represent mixed games as pairs of fixed games obtained by considering all draws to be winning for R or for L respectively, and to work with fixed tensor product and the corresponding linear implication in the single components. We carry out this construction in the full generality offered by our framework, building a symmetric monoidal closed category $\mathcal{Y}_{\mathcal{A}}$, parametric w.r.t. $\mathcal{A}$.

Definition 12. *Let x be a mixed game. We define the pair $\langle x^-, x^+ \rangle$ of fixed games as follows: x^- is obtained from x by taking all infinite plays which are draws on x to be winning for R; x^+ is obtained from x by taking all infinite plays which are draws on x to be winning for L.*

Notice that each mixed game is uniquely determined by its corresponding pair of fixed games. In particular, for fixed games x, we have $x = x^- = x^+$.

Definition 13 (The Category $\mathcal{Y}_{\mathcal{A}}$)
Objects*:* *mixed games $x = \langle x^-, x^+ \rangle$.*
Morphisms*:* *pairs of winning strategies for LII, $\langle \sigma^-, \sigma^+ \rangle : \langle x^-, x^+ \rangle \to \langle y^-, y^+ \rangle$.*

Identities on the category $\mathcal{Y}_{\mathcal{A}}$ are the pairs of *copy-cat strategies*, and closure under composition follows by closure in the single components. Fixed tensor product in the components naturally induces a tensor on the category $\mathcal{Y}_{\mathcal{A}}$. This yields the structure of a symmetric monoidal closed category on $\mathcal{Y}_{\mathcal{A}}$. In particular, we define: $x \otimes y = \langle x^- \otimes y^-, x^+ \otimes y^+ \rangle$, $\overline{x} = \langle \overline{x^-}, \overline{x^+} \rangle$, $x \multimap y = \langle x^- \multimap y^-, x^+ \multimap y^+ \rangle = \langle \overline{\overline{x^- \otimes \overline{y^-}}}, \overline{\overline{x^+ \otimes \overline{y^+}}} \rangle$. Closure of the monoidal category $\mathcal{Y}_{\mathcal{A}}$ follows from the identity $x \otimes y \multimap z = y \multimap (x \multimap z)$. Hence, we have:

Theorem 3. *The category $\mathcal{Y}_{\mathcal{A}}$ is symmetric monoidal closed.*

Notice that $\mathcal{Y}_{\mathcal{A}}$ is *not* $*$-autonomous, since $x \otimes y \multimap \overline{z} \neq x \multimap \overline{(y \otimes z)}$ on mixed games. But, if we restrict $\mathcal{Y}_{\mathcal{A}}$ to fixed games, we get the category $\mathcal{X}_{\mathcal{A}}$, which is $*$-autonomous. Moreover, $\mathcal{Y}_{\mathcal{A}}$ restricted to well-founded games is compact closed, with negation giving dual objects. Namely, the copy-cat strategy induces natural winning strategies for LII on $x \otimes \overline{x} \multimap 0$ and $0 \multimap x \otimes \overline{x}$, where 0 denotes the empty game, for any well-founded game x. Thus we have:

Theorem 4. *The full subcategory of $\mathcal{Y}_{\mathcal{A}}$, consisting of well-founded games and winning strategies, is compact closed.*

As shown in Section 1.1, non-wellfounded Conway games can be represented in our framework by instantiating $\mathcal{A}$ to an appropriate 2-element set. Let us denote by $\mathcal{Y}_2$ the corresponding category. As a corollary of Theorem 3 we get:

Corollary 1. *The category $\mathcal{Y}_2$ of Conway games is symmetric monoidal closed.*

Moreover, by restricting to well-founded games, we obtain Joyal's category as a full subcategory, and from Theorem 4 we have:

Corollary 2 ([Joy77]). *The category of Conway games and winning strategies is compact closed.*

Game Equivalences. Having defined games as a final coalgebra, games are already taken up-to bisimilarity, thus abstracting from superficial features of positions. Bisimilarity is a first structural equivalence on game graphs, but on top of this one can define various equivalences and congruences, by looking at strategies. Such equivalences arise in many conceptually different ways, and they have been studied for Conway games and hypergames in [HL11, HLR11].

Our categorical constructions give rise to interesting notions of (pre)equivalences on games induced by the morphisms, and defined by:

$$x \leq_{\mathcal{Y}_\mathcal{A}} y \quad \text{iff} \quad \text{there exists a winning strategy for LII on } x \multimap y \ .$$

Notice that, since $\mathcal{X}_\mathcal{A}$ is a full subcategory of $\mathcal{Y}_\mathcal{A}$, the (pre)equivalence induced by $\mathcal{X}_\mathcal{A}$ coincides with the restriction on fixed games of $\leq_{\mathcal{Y}_\mathcal{A}}$.

Identities on the category $\mathcal{Y}_\mathcal{A}$ correspond to reflexivity of $\leq_{\mathcal{Y}_\mathcal{A}}$, closure under composition corresponds to transitivity, while functoriality of tensor and negation ensures congruence of $\leq_{\mathcal{Y}_\mathcal{A}}$ w.r.t. the corresponding operations.

Interestingly, the equivalence $\leq_{\mathcal{Y}_2}$ captured by our category $\mathcal{Y}_2$ of mixed Conway games coincides with the loopy game equivalence of [BCG82].

Definition 14 (Loopy Equivalence). *For x, y mixed games, we define:*

$$x \leq_l y \quad \textit{iff} \quad \textit{there are non-losing strategies for LII on } \overline{x^-} \oplus y^- \textit{ and } \overline{x^+} \oplus y^+ .$$

Then we have:

Theorem 5. *For mixed games x, y,* $\quad x \leq_l y \iff x \leq_{\mathcal{Y}_2} y$.

Proof. We prove that $\overline{x^- \otimes \overline{y^-}}$ $(\overline{x^+ \otimes \overline{y^+}})$ has a *winning strategy* for LII iff $\overline{x^- \oplus \overline{y^-}}$ $(\overline{x^+ \oplus \overline{y^+}})$ has a *non-losing strategy* for LII. Equivalently, $x^- \otimes \overline{y^-}$ $(x^+ \otimes \overline{y^+})$ has a *winning strategy* for RII iff $x^- \oplus \overline{y^-}$ $(x^+ \oplus \overline{y^+})$ has a *non-losing strategy* for RII. This follows since, for fixed games, $x \otimes y$ has a winning strategy for R (I or II) iff $x \oplus y$ has a non-losing strategy for R (I or II). □

4 Final Remarks and Directions for Future Work

We have considered a general notion of coalgebraic game, whereby non-wellfounded games are viewed as elements of a final coalgebra. This allows for a unified treatment of games arising in different settings, in particular Conway games and AJM-games, and it helps in shedding light on the relationships between them. We have introduced and studied general notions of categories of games and strategies, subsuming Joyal's category of Conway games as well as categories used in Game Semantics. Categorical equivalences have been defined, providing in particular a characterization of the equivalence on loopy games of [BCG82].

Here is a list of further comments and directions for future work.

Partial strategies. In this paper, we have considered *total* strategies, but in contexts such as Game Semantics, often *partial* strategies are considered. Categories of games and partial strategies in the spirit of Joyal's category have been studied *e.g.* in [HS02, Mel09, MTT09]. Partial strategies could be naturally modeled in our framework. It would be interesting to investigate the relationships between partial and total strategies in generality. Intuitively, partial strategies should allow to approximate total strategies up to plays of certain length.

Exponential. The general categorical constructions carried out in the present paper provide symmetric monoidal closed and $*$-autonomous categories, which allow to model fragments of Linear Logics. We claim that mixed games can be endowed with an exponential operation, endowing $\mathcal{Y}_{\mathcal{A}}$ with a structure of linear category. The exponential operation can be defined by $!x = \Sigma^{\infty} x^R$, where x^R is obtained from x by erasing all L-opening moves, and Σ^{∞} is an "infinite sum" operation, which can be defined via a suitable *generalized coiteration schema.*

Games with payoff on a partially ordered set. In [San02], partial infinite games are introduced. Various operations are defined, including a kind of sum operation. A precise comparison with our categorical sums has still to be investigated.

Generalizing the coalgebraic framework. Coalgebraic games can be further generalized, by encoding more information in the parameter set $\mathcal{A}$, *e.g.* the payoff or the turn of the players. These allow us to model a wider range of games, including automata games and games arising in Economics. It would be also interesting to investigate a generalization for *non-perfect information games.* An approach could be that of explaining them using the notion of coalgebra morphism.

Coinductive specification of strategies. One can give a *coinductive* definition of the set of strategies for a player, via corecursive equations. Intuitively, if we denote by $\mathcal{S}_L(s)$ the set of strategies for L starting on the play s, a strategy in $\mathcal{S}_L(s)$, where s ends with the current position $\langle a, x\rangle$, amounts to: either the empty strategy, if L has no move in the position x; or a strategy for L in $\mathcal{S}_L(s\langle a', x'\rangle)$, where $\langle a', x'\rangle \in x$ and a' is a L move, if L is next to move in the current position $\langle a, x\rangle$, *i.e.* a strategy in $\Sigma_{\langle a',x'\rangle \in x . \lambda a' = L} \mathcal{S}_L(s\langle a', x'\rangle)$; or a collection of strategies for L, for any possible R move from the current position $\langle a, x\rangle$, if R is next to move, *i.e.* a collection of strategies in $\Pi_{\langle a',x'\rangle \in x . \lambda a' = R} \mathcal{S}_L(s\langle a', x'\rangle)$.

It would be interesting to formalize the above idea.

References

[Abr96] Abramsky, S.: Semantics of Interaction: an introduction to Game Semantics. In: Dybjer, P., et al. (eds.) CLiCS 1996 School. Cambridge University Press (1997)

[AJM00] Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Information and Computation 163, 404–470 (2000)

[Acz88] Aczel, P.: Non-wellfounded sets. CSLI Lecture Notes, Stanford, vol. 14 (1988)

[BM96] Barwise, J., Moss, L.: Vicious Circles. CSLI Lecture Notes, Stanford, vol. 60 (1996)

[Ben02] van Benthem, J.: Extensive games as process models. Journal of Logic, Language and Information 11 (2002)

[BCG82] Berlekamp, E., Conway, J., Guy, R.: Winning Ways. Academic Press (1982)

[Con01] Conway, J.H.: On Numbers and Games. A K Peters Ltd. (2001)

[FH83] Forti, M., Honsell, F.: Set-theory with free construction principles. Ann. Scuola Norm. Sup. Pisa, Cl. Sci. 10(4), 493–522 (1983)

[HL11] Honsell, F., Lenisa, M.: Conway Games, algebraically and coalgebraically. Logical Methods in Computer Science 7(3) (2011)

[HLR11] Honsell, F., Lenisa, M., Redamalla, R.: Equivalences and Congruences on Infinite Conway Games. In: Theoretical Informatics and Applications (2012)

[HS02] Hyland, M., Schalk, A.: Games on Graphs and Sequentially Realizable Functionals. In: LICS 2002, pp. 257–264. IEEE Computer Science Press (2002)

[Joy77] Joyal, A.: Remarques sur la Theorie des Jeux a deux personnes. Gazette des Sciences Mathematiques du Quebec 1(4) (1977)

[Mel09] Mellies, P.A.: Categorical semantics of linear logic. In: Panoramas et Synthéses, vol. 27, Société Mathématique de France (2009)

[MTT09] Melliès, P.-A., Tabareau, N., Tasson, C.: An Explicit Formula for the Free Exponential Modality of Linear Logic. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 247–260. Springer, Heidelberg (2009)

[Pau00] Pauly, M.: From Programs to Games: Invariance and Safety for Bisimulation. In: CSL 2009, pp. 485–496 (2009)

[San02] Santocanale, L.: Free μ-lattices. J. Pure Appl. Algebra 168, 227–264 (2002)

Quasi-recognizable vs MSO Definable Languages of One-Dimensional Overlapping Tiles
(Extended Abstract)

David Janin

Université de Bordeaux, LaBRI UMR 5800,
351, cours de la libération,
F-33405 Talence
janin@labri.fr

Abstract. It has been shown [6] that, within the McAlister inverse monoid [10], whose elements can be seen as overlapping one-dimensional tiles, the class of languages recognizable by finite monoids collapses compared with the class of languages definable in Monadic Second Order Logic (MSO).

This paper aims at capturing the expressive power of the MSO definability of languages of tiles by means of a weakening of the notion of algebraic recognizability which we shall refer to as quasi-recognizability. For that purpose, since the collapse of algebraic recognizability is intrinsically linked with the notion of monoid morphism itself, we propose instead to use premorphisms, monotonic mappings on ordered monoids that are only required to be sub-multiplicative with respect to the monoid product, i.e. mapping φ so that for all x and y, $\varphi(xy) \leq \varphi(x)\varphi(y)$.

In doing so, we indeed obtain, with additional but relatively natural closure conditions, the expected quasi-algebraic characterization of MSO definable languages of positive tiles. This result is achieved via the axiomatic definition of an original class of well-behaved ordered monoid so that quasi-recognizability implies MSO definability. An original embedding of any (finite) monoid S into a (finite) well-behaved ordered monoid $\mathcal{Q}(S)$ is then used to prove the converse.

1 Introduction

This paper is rooted in formal language theory where, classically, the object of study are sets (languages) of words (or strings) over a finite alphabet A, that is, subsets of the free monoid A^*. A main topic of research is the characterization of various classes of languages and, for this, there are a number of approaches: recognition by automata of various kinds, recognition by monoids, definability through various logical systems. The vast theoretical corpus that has been developed has inspired the study of languages (subsets) in monoids other than free monoids. A notable example is the theory of trace languages [3] where traces model concurrent behaviors. Another example is the study of subsets of free inverse monoids [17].

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 516–528, 2012.

In this paper, the monoid that replaces the free monoid is the monoid T_A of positive overlapping tiles. This monoid extends the free monoid A^* by adjoining left and right compatibility constraints to words which define whether the concatenation product is legal or not. The resulting objects are triples of words extended with 0 to complete the product. Indeed, the product of incompatible tiles is set to 0.

Our interest in languages of tiles originally stems from computational music theory [8,1] where the monoid of positive tiles is defined as an abstraction of sequential combinations of musical sequences with overlapping anacrusis or conclusions.

It is also possible to generalize these modeling perspectives. Indeed, overlapping tiles can be seen as extended models of *guarded processes*. The left compatibility constraint of a tile can be seen as a sequence of actions that must occur *before* a process is executed. Symmetrically, the right compatibility constraint of a tile can be seen as a sequence of actions that must occur *after* a process is executed. The behavior of a complex sequential process, with both past and future guards for each executions, can thus be modeled as the set of its possible guarded executions. This is a *language* of non zero tiles.

The study of languages of tiles have already been the subject of research on our part [6]. The class of languages of tiles definable in Monadic Second Order Logic (MSO) is shown to be *simple*, i.e. these languages are finite sums of cartesian products of regular languages of words. Since non zero tiles are just triples of words this result is not a surprise. This class is also shown to be *robust*. It is not only closed under boolean operators and projections, but also under (tiles) product, iterated product and left and right residuals. Since the product of tiles involves arbitrarily long pattern matching contraints, this good property was less expected.

Aiming at identifying efficient mechanism to manipulate these languages of tiles, we also have studied recognition by monoids. However, despite interesting mathematical properties related with covers of bi-infinite periodic words [6], the class of languages of tiles recognizable by finite monoids is shown to collapse. It thus provides no interesting notion of automata. Of course, we may try to define, by brute force, finite state machines for MSO definable languages of tiles. However, in order to be truly useful, this notion of machine needs to be compositional with respect to the product of (languages of) tiles. Since compositionality is guaranteed by algebraic approaches, we rather seek to identify ways of remedying the collapse of recognition by monoids.

Within the context of tiles, morphisms, in that they preserve products, convey far too much structure to induce enough expressive power. We thereby seek to identify a *relaxation* of the notion of morphism itself.

In this paper, we introduce the notion of quasi-recognizability: recognizability by means of *premorphisms* instead of morphisms. Defined on *ordered monoids*, premorphisms are mappings that are only required to be sub-multiplicative w.r.t. the monoid product [12], i.e. $\varphi(xy) \leq \varphi(x)\varphi(y)$ instead of $\varphi(xy) = \varphi(x)\varphi(y)$.

But this proposal comes with a price: in general, quasi-recognizability does not imply MSO definability. This means that our proposal is necessarily twofold. On the one hand, we need to find an adequate *restriction* of the category of ordered monoids and premorphisms that preserves MSO definability. On the other, we need the induced subcategory to be as *large* as possible in order to recover our expressiveness yardstick : MSO definability.

Is such a delicate balance achievable? The answer to this question is as yet unknown and may well be negative but we provide here a solution that *essentially* fulfills these goals.

Main Results. We define the subclass of strongly well-behaved ordered monoids and the subclass of well-behaved premorphisms. In the associated category, we prove that all quasi-recognizable languages of tiles are definable, i.e. $QREC \subseteq MSO$, i.e. Theorem 4. Our first goal is achieved. Conversely, provided languages of tiles satisfy a context-coherence closure property (CC), we prove that MSO definable languages of tiles are quasi-recognizable, i.e. $MSO \cap CC \subseteq QREC$, i.e. Theorem 7. Our second goal is *essentially* achieved.

As far as we can see, the context-coherence closure property seems inherently linked with the notion of overlapping tiles itself. Moreover, the class $QREC$ is much larger than the class REC of (classically) recognizable languages of tiles. Indeed, the class $QREC$ contains all (embeddings of) regular word languages. It thus truly generalizes the class of recognizable languages of words. In contrast, as far as the embedding of languages of words is concerned, it can be shown [6] that the classical recognizability over tiles leads to finite boolean combination of languages of words of the form $u(vu)^k(vu)^*$ with $u \in A^*$, $v \in A^+$ and $k \in I\!N$. This class is even smaller than the related case of languages of words defined by finite inverse monoids [11].

Of course, our proposal may seem technical. These are new ideas and methods. Their presentation has not as yet benefited from the fine-tuning that tried and tested practice brings. Setting up an adequate subcategory also requires the definition of a non trivial subclass of ordered monoids. Further still, our proof that context-coherent MSO-definable languages are quasi-recognizable relies on a *original* embedding of any monoid S into a well-behaved ordered monoid $\mathcal{Q}(S)$. The mathematical elegance of this construction justifies, a posteriori, the technicalities that led to its definition.

Related Works. Must we necessarily therefore conclude that our proposal is totally disconnected from former studies?

In mathematics, it results that the (well-behaved) ordered monoids, considered in this paper, generalize the two-sided version of Restriction or Ehresmann monoids [4]. Further still, it results that well-behaved ordered monoids coincide with a stable order restriction of Lawson's U-semiadequate monoids [9].

More precisely, the definitions advanced in this paper are based on ordered monoids. In another paper [7], we set forward an equivalent axiomatic (bi-unary) definition of the class of well-behaved ordered monoids. In doing so, the relationship with other known classes of monoids or semigroups is made clear in this particularly rich field of research of semigroups with local units (see [5]) at the frontier with the even richer inverse semigroup theory. Furthermore the main construction, the embedding of any monoid S into a well-behaved ordered monoid $\mathcal{Q}(S)$, proves to be an *expansion* in the sense of Birget and Rhodes [16] within the category of premorphisms on ordered monoids [7]. As such, it may have many more interesting consequences in that specific mathematical field.

In computer science itself, though more implicitly, the notion of tiles and the related notion of concatenation products has been a subject of research for many years. In his study of two-way automata, Pécuchet [14] defines partial runs's domains as (positive or negative) overlapping tiles. The product of tiles defined in this paper explicits the composition of the underlying domains resulting from the composition of two partial runs. Later, Birget proposes an algebraic reduction of two-way automata into finite monoids [2] where these ideas are further developed. However, despite many attempts since then, the question raised by Birget: *does a true algebraic characterization of two-way automata in fact exist*, remains unanswered so far.

In this paper, we do provide an algebraic setting for languages of tiles. And even the context coherence requirement no longer stands when two special letters mark (as for two-way automata) the beginning and the end of the word upon which given tiles are considered. Thereby, our proposal may constitute a first step towards a positive answer to Birget's long standing question.

Some Notations. For every monoid S, for every x and $y \in S$, we often write xy for the product $x.y$ of x and y in S. By extension, given $X \subseteq S$, and $x \in S$, let xX (resp. Xx) be the set $xX = \{xy \in S : y \in X\}$ (resp. $Xx = \{yx \in S : y \in X\}$).

The prefix preorder $\leq_p$ (resp. the suffix preorder $\leq_s$) is defined, for all x and $y \in S$ by $x \leq_p y$ when $xz = y$ for some $z \in S$ (resp. $x \leq_s y$ when $zx = y$ for some $z \in S$). Under both prefix and suffix preorder, the neutral element 1 is the least element of S, and the absorbant element 0 (if there is such) is the greatest. We also write $x^{-1}(y) = \{z \in S : xz = y\}$ and $(y)x^{-1} = \{z \in S : zx = y\}$. This notation extends to sets as follows: for all $x \in S$ and $Y \subseteq S$ we write $x^{-1}(Y) = \{z \in S : xz \in Y\}$ and $(Y)x^{-1} = \{z \in S : zx \in Y\}$.

The free monoid A^* on the alphabet A is extended with 0 with, for all $u \in A^*$, $u0 = 0u = 0$. Then, for every two words u and v, $u^{-1}(v)$ (resp. $(v)u^{-1}$) is defined as the unique word, if it exists, such that $v = uu^{-1}(v)$ (resp. $v = (v)u^{-1}u$) or 0 otherwise. Last, we write $u \vee_p v$ (resp. $u \vee_s v$) for the smallest word $w \in A^* + 0$ such that both $u \leq_p w$ and $v \leq_p w$ (resp. $u \leq_s w$ and $v \leq_s w$). In other words, $u \vee_s v$ (resp. $u \vee_p v$) equals the greatest word among u or v when they are suffix-comparable (resp. prefix-comparable) or it equals 0 otherwise.

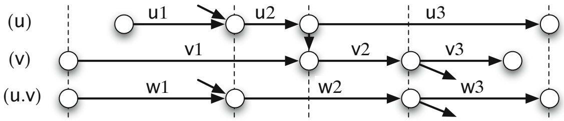
2 Monoids and Languages of Positive Tiles

We review here the definition of the monoid of positive tiles. We also review how it can be seen as an ordered monoid with many additional properties. Our former characterization of MSO definable languages of tiles is then presented [6].

Here, this characterization is used as an alternative and simpler definition of this class of languages of tiles.

Positive Tiles. The set T_A of positive tile on the alphabet A is defined as the set of triples of words $u = (u_1, u_2, u_3) \in A^* \times A^* \times A^*$ extended with an extra tile 0 called the *undefined tile.*

For all non zero tiles $u = (u_1, u_2, u_3)$ and $v = (v_1, v_2, v_3)$, the *sequential product of* u *and* v is defined to be $u.v = ((u_1.u_2 \vee_s v_1)u_2^{-1}, u_2v_2, v_2^{-1}(u_3 \vee_p v_2v_3))$ when both the *left matching constraint* $u_1.u_2 \vee_s v_1 \neq 0$ (u_1u_2 and v_1 are suffix-comparable) and the *right matching constraint* $u_3 \vee_p v_2v_3 \neq 0$ (u_3 and v_2v_3 are prefix-comparable) are satisfied. This definition is illustrated by the following figure where matching constraints are modeled, on the vertical dimension, by letter to letter equality.



The product is completed by $u.v = 0$ when the matching constraint is not satisfied. For instance, with a, b and c three distinct letters, we have $(a, b, c).(b, c, a) = (a, bc, a)$, $(a, b, c).(a, b, c) = 0$ and $(b, 1, ac).(ab, 1, a) = (ab, 1, ac)$.

Set T_A equipped product of tiles is a monoid from now on called the monoid of positive tiles. The neutral element $(1, 1, 1)$ is denoted by 1.

Natural Order. Monoid T_A is conveniently seen as an *ordered monoid* with the natural order relation defined for all $u \in T_A$ by $0 \leq u$ and for all u and $v \in T_A$ with $u = (u_1, u_2, u_3)$ and $v = (v_1, v_2, v_3)$, by $u \leq v$ when $v_1 \leq_s u_1$, $v_2 = u_2$ and $v_3 \leq_p u_3$.

The following lemma, proved in [6], summarizes some properties of the natural order proved. For all non zero tile $u = (u_1, u_2, u_3)$, let $u_L = (u_1u_2, 1, u_3)$ and let $u_R = (u_1, 1, u_2u_3)$. This notation is extended to 0 by setting $0_L = 0_R = 0$.

Lemma 1. *The set* $U(T_A) = \{t \in T_A : t \leq 1\}$ *of* subunits *of* T_A*, is a commutative monoid of idempotent elements and, ordered by the natural order, it is a lattice with product as meet. Moreover, for all* $u \in T_A$, $u_L = \bigwedge\{x \in U(T_A) : ux = u\}$ *and* $u_R = \bigwedge\{x \in U(T_A) : xu = u\}$*, and, for all* $v \in T_A$, $u \leq v$ *if and only if* $u = u_R v u_L$.

The natural order also induces a structure theorem.

Theorem 1. *Monoid* T_A *is completely determined by the submonoid* $\widehat{T_A}$ *of its maximal elements (isomorphic to* A^**) and the lattice* $U(T_A)$ *of its subunits.*

Proof. For all $u \in A^*$, let $u_C = (1, u, 1)$. The mapping $u \mapsto u_C$ from A^* to T_A is a one-to-one morphism. These *canonical images* of words into tiles form the submonoid $\widehat{T_A}$ of maximal elements w.r.t. the natural order of T_A. Staying coherent with notations above, for all $u \in A^*$, let also $u_L = (u, 1, 1) = (u_C)_L$ and $u_R = (1, 1, u) = (u_C)_R$. Both u_L and $u_R \in U(T_A)$. Moreover, for all non zero tile $(u, v, w) \in T_A$ we have $(u, v, w) = u_L v_C w_R$. □

MSO Definable Languages of Tiles. Given a language $L \subseteq T_A - 0$, the *word* congruence $\simeq_L$ associated to L is defined to be the greatest word congruence such that, for all u and $v \in A^*$, $u \simeq_L v$ when for all w_1, w_2 and $w_3 \in A^*$, the following equivalences hold: $(w_1 u w_2, w_3, w_4) \in L \Leftrightarrow (w_1 v w_2, w_3, w_4) \in L$, $(w_1, w_2 u w_3, w_4) \in L \Leftrightarrow (w_1, w_2 v w_3, w_4) \in L$, and $(w_1, w_2, w_3 u w_4) \in L \Leftrightarrow (w_1, w_2, w_3 v w_4) \in L$. In [6], this word congruence is shown to capture, in the following sense, MSO definable languages.

Theorem 2. *For all language $L \subseteq T_A - 0$, $L = \Sigma_{(u,v,w)\in L}[u]_L \times [v]_L \times [w]_L$ with, for all $u \in A^*$, $[u]_L = \{v \in A^* : u \simeq_L v\}$. Moreover, L is MSO-definable if and only if the associated word congruence $\simeq_L$ over A^* is of finite index and thus L equals a finite sum of Cartesian products of regular languages.*

3 Well-Behaved Ordered Monoids and Premorphisms

We provide in this section, the foundation of quasi-recognizability. It is based on the notion of well-behaved ordered monoids - an abstract generalization of the monoid of positive tiles, defined from the class of (stable) ordered monoid with zero [15] - and the notion of premorphisms. However, the adequate definition of quasi-recognizability itself is postponed to the next section.

Well-Behaved Ordered Monoids. Let S be an ordered monoid and let $U(S) = \{x \in S : x \leq 1\}$. Elements of $U(S)$ are called the *subunits* of S. Monoid S is a *well-behaved ordered monoid* when it satisfies the following axioms:

(A0) $0 \in U(S)$, i.e. 0 is a subunit,

(A1) for all x, y and $z \in S$, if $x \leq y$ then $xz \leq yz$ and $zx \leq zy$, i.e. the order relation is *stable* by product,

(A2) for all $x \in U(S)$, $x.x = x$, i.e. *subunits are idempotents*,

(A3) for all x and $y \in S$, if $x \leq y$ then there is e and $f \in U(S)$ such that $x = eyf$, i.e. the monoid order is the two-sided variant of Nambooripad's natural order [13].

(A4) for all $x \in S$, both sets $L_x = \{e \in U(S) : xe = x\}$ and $R_x = \{e \in U(S) : ex = x\}$ have least element $x_L = \bigwedge L_x$ and $x_R = \bigwedge R_x$ with $x_L \in L_x$ and $x_R \in R_x$.

Mappings $x \mapsto x_L$ and $x \mapsto x_R$ from S to $U(S)$ are respectively called the *left* and the *right context operators* on S. One can check that these mappings are projective onto mappings, i.e. for all $x \in U(S)$, $x_L = x_R = x$. As they capture the order relation (see Lemma 2 below), this leads us, in [7], to propose an equivalent axiomatization of well-behaved monoids based on these left and right context operators.

But does there exist any well-behaved monoid? Actually yes, many! In particular, every monoid S can be completed into a well-behaved ordered monoid S^0. Indeed, let S^0, called the *trivial ordered extension* of S, be defined by $S^0 = S + 0$ (with a new zero element) and the order relation defined, for all x and $y \in S^0$, by $x \leq y$ if and only if $x = 0$ or $x = y$. The fact that S^0 is well-behaved is straightforward.

One can also check that the monoid of positive tiles is a well-behaved ordered monoid. The next two Lemmas illustrate how well-behaved monoids generalize the monoid of positive tiles.

Lemma 2. *Let S be a well-behaved ordered monoid. Then* 0 *is the* least element *of S and $U(S)$ is a* submonoid *of S, $U(S)$ ordered by natural order is a* meet semi-lattice *with product as meet, and, in particular, $U(S)$ is a* commutative submonoid*.*

We observe that, in well-behaved ordered monoids, idempotents do not necessarily commute. More precisely, given $E(S)$ the set of idempotents of S, axiom (A2) tells that $U(S) \subseteq E(S)$ but still, it may happen that $xy \neq yx$ for some x and $y \in E(S) - U(S)$. In the theory developed here, the distinction made between subunits and idempotents is essential.

Lemma 3. *Let S be a well-behaved ordered monoid. Then for all x and $y \in S$, $x \leq y$ if and only if there exists $e \in U(S)$ and $f \in U(S)$ such that $x = eyf$ if and only if $x = x_R y x_L$.*

Prehomomorphisms. The following definition is adapted from McAlister and Reilly [12]. Let S and T be two ordered monoids. A mapping $\varphi : S \to T$ is a *premorphism* when $\varphi(0) = 0$, $\varphi(1) = 1$, for all x and $y \in S$, if $x \leq y$ then $\varphi(x) \leq \varphi(y)$ and, for all x and $y \in S$, $\varphi(xy) \leq \varphi(x)\varphi(y)$.

Well-behaved ordered monoids and premorphisms forms a category. Indeed, for every premorphism $\varphi : S \to T$ and $\psi : T \to U$, the mapping $\varphi\psi : S \to U$ defined for all $x \in S$ by $\varphi\psi(x) = \psi(\varphi(x))$ is a premorphism.

As a particular case, a premorphism φ such that $\varphi(xy) < \varphi(x)\varphi(y)$ if and only if $xy = 0$ is called a *trivial premorphism*. These premorphisms are already worth being studied as illustrated by the following lemma.

Lemma 4. *Let A be a finite alphabet and let $L \subseteq A^*$ be a regular language. Let $B = A + \sharp_1 + \sharp_2$ with $\sharp_1$ and $\sharp_2$ two distincts new letters. Let $M = \{(1, \sharp_1 u \sharp_2, 1) \in T_B : u \in L\}$. Then there is a finite monoid S and a trivial premorphism $\psi : T_B \to S^0$ such that $M = \psi^{-1}(\psi(M))$.*

Proof. Since $L \subseteq A^*$ is recognizable so if $L' = \sharp_1 L \sharp_2 \subseteq B^*$. It follows that there is a finite monoid S and a morphism $\varphi : B^* \to S$ such that $L' = \varphi^{-1}(\varphi(L'))$. Let then $\psi : T_B \to S^0$ defined, for all $(u, v, w) \in T_B$ by $\psi((u, v, w)) = \varphi(v)$ when $uvw \in \sharp_1 A^* \sharp_2$ and $\psi((u, v, w)) = 0$ otherwise. Then one can easily check that $M = \psi^{-1}(\varphi(L'))$ hence the claim. □

This Lemma tells us that with pre-images of pre-morphism from monoids of positives tiles to finite well-behaved ordered monoid, words being models as maximal tiles, we indeed generalize classical recognizability. However, we cannot take this as a definition of quasi-recognizability!

Indeed, the weakening of the morphism axiom $\varphi(uv) = \varphi(u)\varphi(v)$ into the premorphism axiom $\varphi(uv) \leq \varphi(u)\varphi(v)$ just breaks many standard and useful properties of morphisms. For instance, $\varphi(S)$ is not in general a submonoid of S since nothing ensures it is closed under product, i.e. $(\varphi(S))^*$ is the submonoid induced by $\varphi(S)$. Even worse, over tiles, MSO definability just fails without extra hypothesis. These extra axioms are proposed in the next section.

4 From Quasi-recognizability to MSO Definability

In this section, within the category of well-behaved monoid and premorphisms, we want to settle our definition of quasi-recognizability in such a way that quasi-recognizability implies MSO definability. This is achieved by paying a special attention to maximal elements and the way premorphisms behave on them.

Strongly Well-Behaved Ordered Monoids. A well-behaved ordered monoid S is said to be *strongly* well-behaved when it satisfies the following additional axioms:

(A5) for all non zero $x \in S$ there is a unique maximal element $\widehat{x} \in S$ such that $x \leq \widehat{x}$, i.e. S is closed in some order theoretical sense,

(A6) for all pairs of non zero elements x and $y \in S$, $\widehat{x}\widehat{y} \neq 0$ and $\widehat{\widehat{x}\widehat{y}} = \widehat{x}\widehat{y}$, i.e. maximal elements form a submonoid.

Theorem 3. *Every strongly well-behaved ordered monoid S is completely determined by the submonoid $\widehat{S}$ of its maximal elements and the semi-lattice $U(S)$ of its subunits.*

Proof. By axiom (A5) and Lemma 3, for all $x \in S$, $x = x_R \widehat{x} x_L$ with both x_R and $x_L \in U(S)$. □

The monoid T_A of positive tiles is strongly well-behaved and Theorem 3 generalizes to strongly well-behaved ordered monoids what Theorem 1 says about monoid T_A.

The behavior of premorphisms on maximal elements is then restricted as follows. A premorphism $\varphi : S \to T$ is a *well-behaved premorphism* when both S and T are strongly well-behaved monoids and the following condition are satisfied:

(P1) for all x and $y \in \widehat{S}$, $\varphi(xy) \in \widehat{T}$,

(P2) for all x, y and $z \in \widehat{S}$, $\varphi(x_L y z_R) = (\varphi(x))_L \varphi(y) (\varphi(z))_R$,

where $\widehat{S}$ (resp. $\widehat{T}$) denotes the set of maximal elements of S (resp. T).

Quasi-Recognizable Languages. We say that $L \subseteq S$ is *quasi-recognizable (QREC)* when there is a well-behaved premorphism $\varphi : S \to M$ with finite M such that $L = \varphi^{-1}(\varphi(L))$.

Theorem 4. *Quasi-recognizable subsets of T_A are definable in MSO, i.e. $QREC \subseteq MSO$.*

Proof. (sketch of) Let $\varphi : T_A \to S$ be a well-behaved premorphism with S finite. It suffices to show that, for all $x \in S$, $\varphi^{-1}(x)$ is MSO-definable. Moreover, we can restrict to non zero elements since we have $\varphi^{-1}(0) = T_A - \bigcup_{x \neq 0} \varphi^{-1}(x)$ and MSO definable languages are closed under finite boolean combination.

Let $(u, v, w) \in T_A$. By Theorem 1, $(u, v, w) = u_L v_C w_R$ with $u_L = (u, 1, 1)$, $v_C = (1, v, 1)$ and $w_R = (1, 1, w)$. It follows, by applying axiom (P2), that $\varphi((u, v, w)) = (\varphi(u_C))_L \varphi(v_C)(\varphi(w_C))_R$. In other words, given $\varphi_C : A^* \to S$ defined, for all $u \in A^*$, by $\varphi_C(u) = \varphi(u_C)$, for all non zero $x \in S$,

$$\varphi^{-1}(x) = \bigcup \{\varphi_C^{-1}(y) \times \varphi_C^{-1}(y') \times \varphi_C^{-1}(y'') \subseteq T_A : (y, y', y'') \in \widehat{S}, x = (y)_L y' (y'')_R\}$$

Since S is finite, this union is finite. Moreover, axiom (P1) ensures that φ_C is a morphism (since $\widehat{T_A} = \{u_C \in T_A : u \in A^*\}$) hence, for all $y \in \widehat{S}$, $\varphi_C^{-1}(y) \subseteq A^*$ is a regular language. We conclude by applying Theorem 2. □

Does the converse of Theorem 4 hold ? In general no. But this comes from a rather welcome property: left and right constraints in tiles are. . . just product constraints. It follows that quasi-recognizable languages satisfy a closure property on left and right constraints that is studied below.

Context Coherence Closure Property. The following lemma tells that quasi-recognizable languages are closed w.r.t. to equivalent left and right constraints.

Lemma 5. *Let $\varphi : T_A \to S$ be a well-behaved premorphism with finite S. For all $x \in S$, for all $(u, v, w) \in \varphi^{-1}(x)$, for all u' and $w' \in A^*$, if $\varphi(u)$ is $\mathcal{L}$-equivalent with $\varphi(u')$ and $\varphi(w)$ is $\mathcal{R}$-equivalent to $\varphi(w')$, then $(u', v, w') \in \varphi^{-1}(x)$.*

Proof. Let us first recall that two elements x and $y \in S$ are $\mathcal{L}$-equivalent (resp. $\mathcal{R}$-equivalent) when both $x \leq_s y$ and $y \leq_s x$ (resp. both $x \leq_p y$ and $y \leq_p x$).

Since $\varphi((u, v, w)) = (\varphi(u_C))_L \varphi(v_C)(\varphi(w_C))_R$, the statement then follows from the easy observation that, for all x and $y \in S$, since S is well-behaved, if x and y are $\mathcal{L}$-equivalent (resp. $\mathcal{R}$-equivalent) then $x_L = y_L$ (resp. $x_R = y_R$). □

Observe that the underlying closure property is rather subtle. Indeed, we still lack of explicit canonical minimal structures (as syntactical monoids) that characterize quasi-recognizable languages of tiles. This means we still do not have a way to define that closure property in a minimal way. However, the word congruence $\simeq_L$ associated to every language of tiles still gives us a canonical definition instead.

A language of tiles $L \subseteq T_A$ is *context-coherent* when, given the induced word congruence $\simeq_L$ associated to L, for all tiles $(u, v, w) \in L$, for all u' and $v' \in A^*$, if u and u' are $\mathcal{L}$-equivalent with respect to $\simeq_L$ and w and w' are $\mathcal{R}$-equivalent with respect to $\simeq_L$, then $(u', v, w') \in L$.

Is this closure property a real loss in expressive power? We have seen in Lemma 4 that *plugs* can be used on words so that $\mathcal{R}$-equivalence on right constraints and $\mathcal{L}$-equivalence on left constraints trivialize in some sense. It follows that, from a modeling perspective, the context-coherence constraint is just a matter of modeling choice! In particular, as already mentioned in the introduction, this is probably enough in order to model the behavior of two-way automata on words where left and right plugs are classically used to mark words' extremities.

5 From MSO-Definability to Quasi-Recognizability

Given a MSO definable language $L \subseteq T_A$, assumed to be context-coherent, we need now to provide a finite strongly well-behaved monoid that quasi-recognizes L.

By Theorem 2, since L is MSO definable, the word congruence $\simeq_L$ associated to L is finite. Our point is then to built from $S = A^*/\simeq_L$ (the finite monoid induced by that congruence) a strongly well-behaved ordered monoid $\mathcal{Q}(S)$ that quasi-recognizes L itself. This well-behaved extension $\mathcal{Q}(S)$ of S, is presented in this section.

We will show that this extension is made in such a way that the monoid of positive tiles T_A itself becomes a submonoid of the well-behaved extension $\mathcal{Q}(A^*)$ of the free monoid A^*. In other words, any morphism $\varphi : A^* \to S$ can be lifted to a well-behaved premorphism from $\mathcal{Q}(\varphi) : \mathcal{Q}(A^*) \to \mathcal{Q}(S)$.

Prefix and Suffix Upper Sets. Let S be a monoid. Let $\mathcal{U}_p(S)$ (resp. $\mathcal{U}_s(S)$) be defined as the set of upward closed subsets of S preordered by $\leq_p$ (resp. $\leq_s$) the prefix (resp. suffix) preorder. More precisely, as S is a monoid hence with $1 \in S$, $\mathcal{U}_p(S)$ (resp. $\mathcal{U}_s(S)$) is the set of subsets $U \subseteq S$ such that $US = U$ (resp. $SU = U$).

For both $t = p$ or $t = s$, elements of $\mathcal{U}_t(S)$ are from now on called *x-upper set*. The set $\mathcal{U}_t(S)$ is turned into a monoid by taking $\cap$ as product. Indeed, the intersection of two t-upper sets is a t-upper set and the neutral (or maximal) element is S itself, and $\emptyset$ is the absorbant (or minimal) element. In semigroup theory, non empty elements of $\mathcal{U}_p(S)$ (resp. $\mathcal{U}_s(S)$) are the right (resp. left) ideals of S.

Lemma 6. *Let S be some monoid and let $x \in S$. Set xS is a p-upper set (resp. Sx a s-upper set) and, for every p-upper set (resp. s-upper set) $U \subseteq S$:*

(1) if $x \in U$ then $x^{-1}(U) = S$ (resp. $(U)x^{-1} = S$),
(2) xU is a p-upper set (resp. Ux is a s-upper set),
(3) $x^{-1}(U)$ is a p-upper set (resp. $(U)x^{-1}$ is a s-upper sets),
(4) $xx^{-1}(U) \subseteq U \subseteq x^{-1}(xU)$ (resp. $(U)x^{-1}x \subseteq U \subseteq (Ux)x^{-1}$),

The Strongly Well-Behaved Extension. Let S be a monoid. The extension $\mathcal{Q}(S)$ of S is defined to be $\mathcal{Q}(S) = (\mathcal{U}_s(S) - \emptyset) \times S \times (\mathcal{U}_p(S) - \emptyset) + 0$ with, for all non zero element $u_1 = (L_1, x_1, R_1)$ and $u_2 = (L_2, x_2, R_2)$ the product $u_1 u_2$ defined by $u_1.u_2 = (L_1 \cap (L_2)x_1^{-1}, x_1x_2, R_2 \cap x_2^{-1}(R_1))$ when both compatibility contraints $L_1 \cap (L_2)x_1^{-1} \neq \emptyset$ and $R_2 \cap x_2^{-1}(R_1) \neq \emptyset$ are satisfied, and by $u_1.u_2 = 0$ otherwise.

The expected natural order is defined as follows. For all pairs of non zero elements $u_1 = (L_1, x_1, R_1)$ and $u_2 = (L_2, x_2, R_2) \in \mathcal{Q}(S)$, we say that $u_1 \leq u_2$ when $L_1 \subseteq L_2$, $x_1 = x_2$ and $R_1 \subseteq R_2$. This relation is extended to zero by taking $0 \leq u$ for all $u \in \mathcal{Q}(S)$.

As an ordered monoid, we already have:

Lemma 7. *The mapping $i : S^0 \to \mathcal{Q}(S)$ that maps zero to zero and any non zero element $x \in S$ to $i(x) = (S, x, S)$ is a one-to-one monoid morphism. The set $i(S)$, isomorphic to S, is the submonoid of $\mathcal{Q}(S)$ that contains exactly all maximal elements of $\mathcal{Q}(S)$. The mapping $\pi : \mathcal{Q}(S) \to S^0$ that maps 0 to 0 and any non zero element (L, x, R) to $\pi(L, x, R) = x$ is an onto trivial premorphism with $\pi \circ i = I_{S^0}$, the identity mapping in S^0.*

Proof. The fact that i is a monoid morphism is immediate and the product on elements of $i(S)$, all of the form (S, x, S), just mimics the product in S since, for all $x \in S$, $x^{-1}(S) = (S)x^{-1} = S$.

Mapping π is obviously onto since $\pi \circ i = I_{S^0}$. Then we check that it is a (trivial) premorphism, i.e. for all u_1 and $u_2 \in \mathcal{Q}(S)$ either $u_1 u_2 \neq 0$ and then $\pi(u_1 u_2) = \pi_S(u_1)\pi_S(u_2)$ or $u_1 u_2 = 0$ and thus $\pi(u_1 u_2) = 0$. □

Moreover, as intended:

Theorem 5. *For all monoid S, the monoid $\mathcal{Q}(S)$ ordered by the extension order $\leq$ is a strongly well-behaved ordered monoid.*

Last, the following theorem says that our construction above essentially extends to arbitrary monoids the way the monoid of positive tiles is built from the free monoid A^*.

Theorem 6. *There is a one to one morphism $i : T_A \to \mathcal{Q}(A^*)$ such that, for all $u \in T_A$, $i(u_L) = (i(u))_L$ and $i(u_R) = (i(u))_R$.*

Proof. Observe that A^* is totally ordered by $\leq_s$ and $\leq_p$. It follows that for $x = p$ and $x = s$, the mapping $\varphi_x : A^* + 0 \to \mathcal{U}_x(A^*)$ defined, for every $u \in A^*$, by $\varphi_x(u) = \{v \in A^* : u \leq_x v\}$ is one-to-one. It is then an easy task to check that $i : T_A \to \mathcal{Q}(A^*)$ defined by $i(0) = 0$ and, for every tile $(u, v, w) \in A_T$, $i((u, v, w)) = (\varphi_s(u), v, \varphi_p(w))$ is a one-to-one morphism. The last property is immediate. □

From MSO-Definability to Quasi-Recognizability. We are now ready to make the picture complete. More precisely:

Theorem 7. *If $L \subseteq T_A$ is MSO definable and context coherent then L is quasi-recognizable.*

Proof. Let $L \subseteq T_A$ be an MSO definable language of positive tiles and let $S = A^*/\simeq_L$ be the finite monoid defined by the quotient of A^* under the (finite index) word congruence induced by L. For every word $u \in A^*$, let us write $[u] \in S$ the class of words of A^* equivalent under $\simeq_L$ to u.

Let define $\varphi : T_A \to \mathcal{Q}(S)$ by taking, for every non zero positive tile $u = (u_1, u_2, u_3) \in T_A$, $\varphi(u) = (S[u_1], [u_2], [u_3]S)$. One can check that φ is a well-behaved premorphism. Moreover, by construction, $L \subseteq \varphi^{-1}(\varphi(L))$.

The converse inclusion follows from the fact that L is context coherent. □

6 Conclusion

We have defined and studied the notion of quasi-recognizability: algebraic recognizability by means of premorphism instead of morphisms. Applied to the monoid of positive tiles, this notion is shown to be effective. It is also a successful remedy to the collapse of classical recognizability by monoid morphisms.

Now, the potential link with two-way automata needs to be further investigated. In particular, studying quasi-recognizability over arbitrary (positive and negative) tiles seems necessary. The study of quasi-recognizability of subsets of the free inverse monoid, which elements are birooted trees, may also be of interest. It may lead to a new algebraic approach to regular languages of trees.

References

1. Berthaut, F., Janin, D., Martin, B.: Advanced synchronization of audio or symbolic musical patterns. Technical Report RR1461-12, LaBRI, Université de Bordeaux (2012)
2. Birget, J.-C.: Concatenation of inputs in a two-way automaton. Theoretical Computer Science 63(2), 141–156 (1989)
3. Diekert, V.: The Book of Traces. World Scientific Publishing Co., Inc., River Edge (1995)
4. Gould, V.: Restriction and Ehresmann semigroups. In: Proceedings of the International Conference on Algebra 2010. World Scientific (2010)
5. Hollings, C.D.: From right PP monoids to restriction semigroups: a survey. European Journal of Pure and Applied Mathematics 2(1), 21–57 (2009)
6. Janin, D.: On languages of one-dimensional overlapping tiles. Technical Report RR-1457-12, LaBRI, Université de Bordeaux (2012)
7. Janin, D.: Quasi-inverse monoids. Technical Report RR-1459-12, LaBRI, Université de Bordeaux (2012)
8. Janin, D.: Vers une modélisation combinatoire des structures rythmiques simples de la musique. Revue Francophone d'Informatique Musicale 2 (to appear, 2012)
9. Lawson, M.V.: Semigroups and ordered categories. i. the reduced case. Journal of Algebra 141(2), 422–462 (1991)
10. Lawson, M.V.: McAlister semigroups. Journal of Algebra 202(1), 276–294 (1998)
11. Margolis, S.W., Pin, J.-E.: Languages and Inverse Semigroups. In: Paredaens, J. (ed.) ICALP 1984. LNCS, vol. 172, pp. 337–346. Springer, Heidelberg (1984)
12. McAlister, D.B., Reilly, N.R.: E-unitary covers for inverse semigroups. Pacific Journal of Mathematics 68, 178–206 (1977)
13. Nambooripad, K.S.S.: The natural partial order on a regular semigroup. Proc. Edinburgh Math. Soc. 23, 249–260 (1980)

14. Pécuchet, J.-P.: Automates boustrophedon, semi-groupe de birget et monoide inversif libre. ITA 19(1), 71–100 (1985)
15. Pin, J.-E.: Mathematical foundations of automata theory. Lecture Notes (2011)
16. Rhodes, J., Birget, J.-C.: Almost finite expansions of arbitrary semigroups. J. Pure and Appl. Algebra 32, 239–287 (1984)
17. Silva, P.V.: On free inverse monoid languages. ITA 30(4), 349–378 (1996)

An Improved Approximation Scheme for Variable-Sized Bin Packing*

Klaus Jansen and Stefan Kraft

Department of Computer Science, Theory of Parallelism,
Christian-Albrechts-University to Kiel, 24098 Kiel, Germany
{kj,stkr}@informatik.uni-kiel.de

Abstract. The variable-sized bin packing problem (VBP) is a well-known generalization of the NP-hard bin packing problem (BP) where the items can be packed in bins of M given sizes. The objective is to minimize the total capacity of the bins used. We present an AFPTAS for VBP and BP with performance guarantee $P(I) \leq (1+\varepsilon)OPT(I)+O(\log^2(\frac{1}{\varepsilon}))$. The additive term is much smaller than the additive term of already known AFPTAS. The running time of the algorithm is $O(\frac{1}{\varepsilon^6}\log\left(\frac{1}{\varepsilon}\right)+\log\left(\frac{1}{\varepsilon}\right)n)$ for bin packing and $O(\frac{1}{\varepsilon^7}\log^2\left(\frac{1}{\varepsilon}\right)+\log\left(\frac{1}{\varepsilon}\right)(M+n))$ for variable-sized bin packing, which is an improvement to previously known algorithms.

Keywords: Bin Packing, Variable-Sized Bin Packing, AFPTAS, Asymptotic Fully Polynomial Approximation Scheme, Integrality Gap.

An instance (I,C) of the variable-sized bin packing problem (VBP) is a pair consisting of a list $I=\{a_1,\ldots,a_n\}$ of items and a list $C=\{c_1,\ldots,c_M\}$ of different bin sizes with $n,M\in\mathbb{N}$. Every item $a\in I$ has a size $s(a)\in\,]0,1]$. The bin sizes $c_l\in C$ satisfy $c_l\in\,]0,1]$, and there is one unit-sized bin $c_M=1$. A set of items $S\subset I$ can be packed in a bin of size c_l, $1\leq l\leq M$, as long as the total volume of the items does not exceed the capacity (i.e. size) of a bin, i.e. $\sum_{a\in S}s(a)\leq c_l$.

The Variable-Sized Bin Packing Problem. *Pack the items I of an instance (I,C) into bins of size in C so that the total size of the bins used is minimized.*

The classic bin packing problem (BP) is obviously a special case of the variable-sized bin packing problem (VBP) with $C=\{1\}$. Closely related to BP and VBP is the (multiple-length) cutting-stock problem (MLCSP), where the input is provided in a more compact form: it consists of a vector (d,M,a,b,c,ρ) of d item sizes $a=(a_1,\ldots,a_d)$, the associated number of items b_i of size a_i, the M stock-lengths $c=(c_1,\ldots,c_M)$ and stock-length prices $\rho=(\rho_1,\ldots,\rho_M)$. It is asked to partition the items into sets so that every set fits into a stock and the total price of stocks used is minimized. In our case, the price of a stock would be equal to its length.

* Research supported by DFG project JA612/14-1, "Entwicklung und Analyse von effizienten polynomiellen Approximationsschemata für Scheduling- und verwandte Optimierungsprobleme".

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 529–541, 2012.

1 Introduction

Known Results. Bin Packing is a classic NP-complete problem [9]. The first to consider it in the form of the cutting-stock problem (but naming it differently) were Kantorovich [15] and Eisemann [5]. Several approximation algorithms with polynomial running time are known for Bin Packing (e.g. First-Fit (FF), Next-Fit (NF), Best-Fit, First-Fit Decreasing (FFD) or Next-Fit Decreasing (NFD)). There is however a lower bound for the absolute approximation ratio $\frac{A(I)}{OPT(I)}$, where I is a BP instance, $OPT(I)$ denotes its optimum and $A(I)$ stands for the number of bins a polynomial-time algorithm A needs to pack I. No efficient algorithm can achieve $\frac{A(I)}{OPT(I)} < \frac{3}{2}$ for all bin packing instances I, unless P = NP [9]. (In fact, First-Fit Decreasing attains this absolute ratio [24].) It is therefore a good idea to consider the asymptotic approximation ratio $\limsup_{k\to\infty} \sup\{\frac{A(I)}{OPT(I)} | OPT(I) = k\}$. For example, every packing $FFD(I)$ found by FFD satisfies $FFD(I) \leq \frac{11}{9}OPT(I) + \frac{6}{9}$ [4] and has therefore an asymptotic approximation ratio of $\frac{11}{9}$, which is obviously smaller than $\frac{3}{2}$.

In 1981, Fernandez de la Vega and Lueker [7] presented an asymptotic polynomial time approximation scheme (APTAS), i.e. an algorithm with asymptotic approximation ratio $(1+\varepsilon)$ for $\varepsilon > 0$. The running time is polynomial in the input size $|I| = n$, but exponential in $\frac{1}{\varepsilon}$. One year later, Karmarkar and Karp [16] found an algorithm satisfying $A(I) \leq (1+\varepsilon)OPT(I)+O(\frac{1}{\varepsilon^2})$ and with polynomial running time in n and $\frac{1}{\varepsilon}$, i.e. an asymptotic fully polynomial time approximation scheme (AFPTAS). In 1991, Plotkin et al. [19] presented an improved algorithm satisfying $A(I) \leq (1+\varepsilon)OPT(I) + O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))$ and with better running time $O(n\log(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon^6}\log^6\frac{1}{\varepsilon})$. Shachnai and Yehezkely [22] reduced the running time further to $O(n\log(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon^4}\log^3(\frac{1}{\varepsilon}) \cdot \min\{\frac{1}{\varepsilon^2}, \frac{1}{\varepsilon^{0.62}}\log^{0.62}(\frac{1}{\varepsilon})N\})$, where N is the longest binary representation of any input element. For general BP instances, the algorithm therefore needs $O(\frac{1}{\varepsilon^6}\log^3(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon})n)$ time. The algorithm is an application of Shachnai's and Yehezkely's AFPTAS for bin packing with size preserving fragmentation (BP-SPF).

The Variable-Sized Bin Packing Problem was investigated by Friesen and Langston [8] who analyzed three algorithms with approximation ratios 2, $\frac{3}{2}$ and $\frac{4}{3}$. Murgolo [18] presented an AFPTAS with performance guarantee $(1+\varepsilon)OPT(I) + O(\frac{1}{\varepsilon^4})$. This was improved to $(1+\varepsilon)OPT(I) + O(\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$ by Shachnai and Yehezkely [22], again by an application of their BP-SPF algorithm. The improved running time of the algorithm is $O((M+n)\log(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon^8}\log^3(\frac{1}{\varepsilon}) \cdot \min\{\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}), \frac{1}{\varepsilon^{0.24}}N\})$, where N is (again) the longest binary representation of any input element. For general instances, the running time is therefore bounded by $O(\frac{1}{\varepsilon^{10}}\log^4(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon})(M+n))$.

Our Result. We have found an AFPTAS for BP and VBP with the improved additive term $O(\log^2(\frac{1}{\varepsilon}))$ and a further improved running time:

Theorem 1. *Let (I,C) be a VBP instance with M bin sizes, and let $OPT(I,C)$ denote its optimal objective value. For a given $0 < \varepsilon \leq \frac{1}{2}$, there is an algorithm P that finds a solution of the instance with objective value of at most*

$P(I,C) \leq (1+\varepsilon)OPT(I,C) + O\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)$. The running time of P is bounded by $O\left(\frac{1}{\varepsilon^7}\log^2\left(\frac{1}{\varepsilon}\right) + \log\left(\frac{1}{\varepsilon}\right)(M+n)\right)$.

Corollary 2. *Let I be a BP instance. There is an AFPTAS algorithm P for BP that finds for $\varepsilon > 0$ a packing of I in $P(I) \leq (1+\varepsilon)OPT(I) + O(\log^2(\frac{1}{\varepsilon}))$ bins, where $OPT(I)$ denotes the optimum for I. The running time is bounded by $O(\frac{1}{\varepsilon^6}\log(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon})n)$.*

Techniques. The basic scheme of our algorithm is similar to the one of other AFPTAS for BP [7,16,19] and VBP [18,22]. Based on a threshold defined by ε, we divide the items I into large and small items I_{large} and I_{small}. The large items are further grouped into sets G_1 and $I_{\text{large}} \setminus G_1$. The set G_1 contains the largest items so that rounding up $I_{\text{large}} \setminus G_1$ results in a set $I^{(1)}$ that still satisfies $OPT(I^{(1)}, C) \leq OPT(I, C)$. To reduce the running time, only a subset $C^{(1)} \subset C$ of bin sizes is used, which does not increase the approximation ratio too much.

The main part is finding a packing of $(I^{(1)}, C^{(1)})$ close to the optimum. For this, the VBP for $(I^{(1)}, C^{(1)})$ is expressed as an integer linear program (ILP), which is a well-known technique (see [5,15,18]):

$$\min c^T x \text{ with } Ax \geq b,\ x \in \mathbb{Z}^q \text{ and } x \geq 0\ . \tag{1}$$

AFPTAS usually consider the relaxed version of (1) with $x \in \mathbb{R}^q_{\geq 0}$. The main difficulty is to solve it efficiently and to round it to an integer solution close to the optimum. Our algorithm does this by combining a method based on the algorithm by Grigoriadis et al. [12] and a practical application of a theoretical result by Shmonin [20,23], which are both adapted to VBP.

Shmonin proved the integrality gap $OPT(I) \leq LIN(I) + O(\log^2 d)$ for cutting stock. Here, $OPT(I)$ stands for the optimal integer value and $LIN(I)$ for the optimal fractional value of the relaxed linear program (1) of a cutting stock instance I. The basic idea of Shmonin's proof is to take an optimal fractional solution x of (1) that is rounded down to the next integer. Some of the items not packed by $\lfloor x \rfloor$ are then packed using NF whereas the other remaining items are rounded using geometric grouping. They form a new VBP instance $\tilde{I}$. This procedure is applied to $\tilde{I}$ and iterated until all items of I have been packed.

In contrast to Shmonin's proof, we solve the LPs only approximately and not optimally by applying to VBP the algorithm by Grigoriadis et al. [12]. The columns of A needed are generated dynamically by solving unbounded knapsack problems. Since the relaxed LPs (1) are only solved approximately, we have to prove that the number of item sizes and the size $Area(I^{(k)})$ of the items halves in every iteration k to bound the final objective value. Note that Shmonin's method is a modification of a bin packing algorithm by Karmarkar and Karp [16]. Moreover, a column generation approach is a common technique [10,11,16,18,19,22]. It is necessary because we normally have an exponential number of columns, i.e. $q \in 2^{O(\frac{1}{\varepsilon}\log^2(\frac{1}{\varepsilon}))}$.

After having packed $I^{(1)}$ (and therefore $I_{\text{large}} \setminus G_1$) as explained above, the remaining items in G_1 and I_{small} can be packed greedily without increasing the approximation ratio too much.

Due to space constraints, some proofs are omitted and can be found in the full version of the paper.

2 The Basic Algorithm

In this section, we will introduce the sub-algorithm $Alg(I^{(1)}, C^{(1)}, \varepsilon)$, which is the core element of our algorithm P.

2.1 Integer Linear Programs and Linear Programs for VBP

Let (I, C) be an instance with d different item sizes and where every item has size at least $s(a) \geq \delta > 0$. We introduce configurations: a configuration $K_j^{(l)}$ for the bin size c_l is a subset $J \subseteq I$ such that the items in J fit into a bin of size c_l, i.e. $\sum_{a \in J} s(a) \leq c_l$. Let $K_1^{(l)}, \ldots, K_{q(\delta,d,l)}^{(l)}$ be all configurations of a bin size c_l. The number $q(\delta, d, l)$ of the configurations is normally exponential in the number of item sizes d.

Let $b_1 > \ldots > b_d$ be the subsequence consisting of the different item sizes in $s_1 \geq \ldots \geq s_n$. Then, a configuration $K_j^{(l)}$ can be described by a multiset $\{a(K_j^{(l)}, b_1) : b_1, \ldots, a(K_j^{(l)}, b_d) : b_d\}$ where $a(K_j^{(l)}, b_i)$ denotes the number of items of size b_i in configuration $K_j^{(l)}$. Furthermore, let n_i be the total number of items of size b_i in I. (Obviously, $n_1 + \cdots + n_d = n$ holds.) It is possible to describe the VBP instance as an integer linear program (ILP):

$$\begin{aligned} &\min \sum_{l=1}^{M} \sum_{j=1}^{q(\delta,d,l)} c_l \; v_j^{(l)} \\ &\sum_{l=1}^{M} \sum_{j=1}^{q(\delta,d,l)} a(K_j^{(l)}, b_i) \; v_j^{(l)} = n_i \qquad \text{for } i = 1, \ldots, d \\ &v_j^{(l)} \in \mathbb{N} \cup \{0\} \qquad\qquad \text{for } l = 1, \ldots, M \text{ and } j = 1, \ldots, q(\delta, d, l) \end{aligned} \tag{ILP-VBP}$$

The optimal value of this ILP is equal to $OPT(I, C)$. We now consider the relaxed linear program (LP) with optimum $LIN(I, C)$ by replacing the condition $v_j^{(l)} \in \mathbb{N} \cup \{0\}$ by $v_j^{(l)} \geq 0$ and the conditions $\ldots = n_i$ by $\ldots \geq n_i$. In the ILP as well as the relaxed LP, a variable $v_j^{(l)}$ can be interpreted as a vertical slice of a bin, packed according to configuration $K_j^{(l)}$. The slice has width $v_j^{(l)}$.

We introduce a partial order on VBP instances that will be useful later. It is a natural extension of the order by Fernandez de la Vega and Lueker [7].

Definition 3. *Let (J_1, C) and (J_2, C) be two VBP instances with the same set of bin sizes C. We write $(J_2, C) \leq (J_1, C)$ if there is an injective function $f : J_2 \to J_1$ such that $s(a) \leq s(f(a))$ for every $a \in J_2$. We write $J_2 \leq J_1$ for item sets if the same condition holds.*

The following lemmas will be used later. Lemma 5 is proved in [7].

Lemma 4. *Let* $Area(I) := \sum_{a \in I} s(a)$ *be the volume of the items in* I*. Then we have* $Area(I) \leq LIN(I, C) \leq OPT(I, C)$*.*

Lemma 5. *If* $J_2 \leq J_1$ *or* $(J_2, C) \leq (J_1, C)$ *holds, then* $Area(J_2) \leq Area(J_1)$*,* $LIN(J_2, C) \leq LIN(J_1, C)$ *and* $OPT(J_2, C) \leq OPT(J_1, C)$*.*

2.2 The Algorithm

Let $(I^{(1)}, C^{(1)})$ be a VBP instance with d_1 item sizes, M_1 bin sizes and with $s(a) \geq \delta$ for all items $a \in I^{(1)}$. The basic algorithm *Alg* is presented below: as mentioned in the introduction, it is an adaptation and a practical application of the methods presented by Karmarkar and Karp [16] and Shmonin [20], [23, Ch. 6], combined with a method based on the algorithm by Grigoriadis et al. [12].

```
    Algorithm $Alg(I^{(1)}, C^{(1)}, \varepsilon)$
(1) Set $k := 1$.
(2) while true do begin
  (2.1) Solve the relaxed linear program corresponding to $(I^{(k)}, C^{(1)})$ approximately with accuracy $(1 + \varepsilon)$ (see Subsection 2.3).
  (2.2) Take the integral part $\lfloor v^{(k)} \rfloor = (\lfloor v_j^{(l,k)} \rfloor)$ of the approximate solution $v^{(k)} = (v_j^{(l,k)})$. Pack the items according to $\lfloor v^{(k)} \rfloor$ in the respective bins: these items are the instance $(I_{\text{int}}^{(k)}, C^{(1)})$. The remaining, non-packed items are the residual instance $(I_{\text{res}}^{(k)}, C^{(1)})$.
  (2.3) if $I_{\text{res}}^{(k)} = \emptyset$ then break; else Transform $(I_{\text{res}}^{(k)}, C^{(1)})$ into two instances $(J'_k, C^{(1)})$ and $(J_k, C^{(1)})$, where the items in $J_k$ have been rounded up (see Subsection 2.4).
  (2.4) Pack $J'_k$ into unit-sized bins with Next-Fit; open a new bin if necessary
  (2.5) if $J_k = \emptyset$ then break; else Set $(I^{(k+1)}, C^{(1)}) := (J_k, C^{(1)})$; $k := k + 1$;
    end
(3) Replace the rounded-up sizes of the items by their original sizes in $I^{(1)}$.
```

Remark 6. The algorithm finishes if either all items in an instance $I^{(k)}$ are packed by $\lfloor v^{(k)} \rfloor$, i.e. $I_{\text{int}}^{(k)} = I^{(k)}$, or all items that have not been packed by $\lfloor v^{(k)} \rfloor$ are contained in J'_k and packed with Next-Fit. If neither of these conditions is met, the remaining unpacked items J_k become the new instance $(I^{(k+1)}, C^{(1)})$, which is processed again in the same way. Since items are packed integrally either by $\lfloor v_j^{(l,k)} \rfloor$ or by Next-Fit, we obtain an integral and feasible packing.

2.3 Solving the LPs Approximately

The algorithm has to approximately solve the relaxation of the integer programs (ILP-VBP) of the instances $(I^{(k)}, C^{(1)})$, where we have d_k different item sizes and M_1 bin sizes. There is a well-known method for LPs of packing problems (see e.g. [13,3,1] for Strip Packing) based on the algorithm by Grigoriadis et al. [12]. The method can be adapted to VBP.

The idea is to first assume $r = LIN(I^{(k)}, C^{(1)}) = 1$ and to solve for $r = 1$ an equivalent max-min resource sharing problem with the algorithm by Grigoriadis et al. [12] (see also [13]). The columns of (ILP-VBP) that are needed can be generated dynamically by approximately solving unbounded knapsack problems

$$\max \sum_{i=1}^{d_k} \frac{p_i}{n_i c_l} z_i \quad \text{s.t.} \quad \sum_{i=1}^{d_k} b_i z_i \leq c_l, z_i \in \mathbb{N} \cup \{0\}$$

for every bin size $c_l, l = 1, \ldots, M_1$. Since normally $r = 1 < LIN(I^{(k)}, C^{(1)})$, we will usually find an infeasible solution not packing all items. However, we can scale the solution for $r = 1$ to get a feasible solution $v = (v_j^{(l)})$ whose value is close enough to the optimum $LIN(I^{(k)}, C^{(1)})$. Moreover, the number of variables $v_j^{(l)} > 0$ can be reduced to $d_k + 1$ without changing the value of the solution [2]. Details and the proof of Th. 7 can be found in the full version of the paper.

Theorem 7. *Let $\varepsilon > 0$, $\bar{\varepsilon} := \frac{\varepsilon}{4}$, and let $(I^{(k)}, C^{(1)})$ be a VBP instance with M_1 bin sizes and d_k item sizes. Moreover, let KP be an approximation algorithm for the unbounded knapsack problem that solves an unbounded knapsack instance with d_k item sizes and accuracy $(1 - \frac{\bar{\varepsilon}}{6})$ within $KP(d_k, \frac{\bar{\varepsilon}}{6})$. There is an algorithm that finds a solution $v = (v_j^{(l)})$ of the relaxed (ILP-VBP) with only d_k+1 variables $v_j^{(l)} > 0$ and satisfying $\sum_l c_l \sum_j v_j^{(l)} \leq (1+\varepsilon) LIN(I^{(k)}, C^{(1)})$. The running time is*

$$O\Bigg(d_k \left(\log(d_k) + \frac{1}{\varepsilon^2}\right) \max\left\{M_1 \cdot KP\left(d_k, \frac{\bar{\varepsilon}}{6}\right), O\left(d_k \log\log\left(d_k \frac{1}{\varepsilon}\right)\right)\right\} + d_k^{2.594}\left(\log d_k + \frac{1}{\varepsilon^2}\right)\Bigg) .$$

2.4 Transforming $I_{\text{res}}^{(k)}$

The transformation of $I_{\text{res}}^{(k)}$ into J_k and J'_k is the geometric rounding technique by Shmonin [20] (see also [23, Ch. 6.4]), which is the core element of his proof $OPT(I) \leq LIN(I) + O(\log^2(d))$ for MLCSP. The rounding is a slight modification of the geometric grouping by Karmarkar and Karp [16]. We therefore refer to [23, Ch. 6.4] for details, or the full version of the paper, and only present the basic idea.

Roughly speaking, J_k contains rounded-up items so that it has less item sizes than $I_{\text{res}}^{(k)}$, and J'_k contains the largest and smallest items of $I_{\text{res}}^{(k)}$ so that $J_k \leq I_{\text{res}}^{(k)}$ holds even with the rounded-up items in J_k. More formally, the items of $I_{\text{res}}^{(k)}$ are partitioned into groups $G_1^{(k)} \cup \ldots \cup G_{K_k}^{(k)}$ of size at least 2 so that $K_k \leq \lfloor \frac{Area(I_{\text{res}}^{(k)})}{2} \rfloor + 1$, and where $G_1^{(k)}$ contains the largest items of $I_{\text{res}}^{(k)}$, $G_2^{(k)}$ the remaining largest items, and so on. Then, we split $G_l^{(k)} = G_l'^{(k)} \cup \Delta G_l^{(k)}$ for $l = 2, \ldots, K_k - 1$ so that $|G_l'^{(k)}| = |G_{l-1}^{(k)}|$. Since $G_{l-1}^{(k)}$ does not contain less items than $G_l'^{(k)}$ for $l = 2, \ldots, K_k - 1$, and these items are not smaller than the largest item in $G_l'^{(k)}$, we can round the items in $G_l'^{(k)}$ up to the largest item in it to obtain $H_l^{(k)}$ where $H_l^{(k)} \leq G_{l-1}^{(k)}$ still holds. We define $J_k := \bigcup_{l=2}^{K_k-1} H_l^{(k)}$ and $J_k' := G_1^{(k)} \cup G_{K_k}^{(k)} \cup \bigcup_{l=2}^{K_k-1} \Delta G_l^{(k)}$.

Lemma 8. *Let d_k be the number of different item sizes that instance $I^{(k)}$ has in the execution of $Alg(I^{(1)}, C^{(1)}, \varepsilon)$, and let k_0 be the number of times the main loop of Alg is executed. The following properties hold:*

- $(J_k, C^{(1)}) \leq (I_{\text{res}}^{(k)}, C^{(1)}) \leq (J_k \cup J_k', C^{(1)})$ *for* $k = 1, \dots, k_0$.
- *The latest moment Alg terminates is when* $Area(I_{\text{res}}^{(k)}) \leq 4$ *because* $I_{\text{res}}^{(k)} = J_k'$ *holds then.*
- $I^{(k+1)} = J_k$ *has* $d_{k+1} \leq K_k - 1 \leq \lfloor \frac{Area(I_{\text{res}}^{(k)})}{2} \rfloor - 1$ *different item sizes for* $k = 1, \dots, k_0 - 1$ *because every* $H_l^{(k)}$ *consists of rounded items of only one size .*

2.5 Analysis and Running Time of the Basic Algorithm

In this section, we prove that $Alg(I^{(1)}, C^{(1)}, \varepsilon)$ finds a packing for a given instance $(I^{(1)}, C^{(1)})$ that is not too far from the optimum $OPT(I^{(1)}, C^{(1)})$.

Theorem 9. *Let $(I^{(1)}, C^{(1)})$ be a VBP instance with $s(a) \geq \delta > 0$ for $a \in I^{(1)}$ and d_1 different item sizes. $Alg(I^{(1)}, C^{(1)}, \varepsilon)$ finds a packing $v = (v_j^{(l)})$ such that $\sum_{l=1}^{M} c_l \sum_j v_j^{(l)} \leq (1 + 4\varepsilon)\, OPT(I^{(1)}, C^{(1)}) + O(\log(\frac{1}{\delta}) \log(d_1))$.*

We need some lemmas. The proof of Lemma 12 is an adaptation of the corresponding proof in [20,23].

Lemma 10. *Let d_k be the number of different item sizes in $I^{(k)}$ and k_0 the number of iterations of the main loop of Alg. Then we have*

1. $Area(I_{\text{res}}^{(k)}) \leq d_k + 1$,
2. $Area(I_{\text{res}}^{(k+1)}) \leq d_{k+1} + 1 \leq \frac{1}{2} Area(I_{\text{res}}^{(k)}) \leq \frac{d_k}{2} + 1$ *for* $k = 1, \dots, k_0 - 1$,
3. $d_k \leq \frac{d_1}{2^k}$ *for* $k = 1, \dots, k_0$, *and*
4. $Area(I_{\text{res}}^{(k)}) \leq \frac{Area(I_{\text{res}}^{(1)})}{2^{k-1}}$ *for* $k = 1, \dots, k_0$ *and* $Area(I^{(k)}) \leq \frac{Area(I^{(1)})}{2^{k-2}}$ *for* $k = 2, \dots, k_0$.

Proof. Let $v^{(k)} = (v_j^{(l,k)})$ be the approximate solution to the relaxed LP (ILP-VBP) corresponding to $I^{(k)}$. As stated in Theorem 7, there are at most $d_k + 1$ many variables $v_j^{(l,k)} > 0$. Thus, there are at most $d_k + 1$ many $v_j^{(l,k)} - \lfloor v_j^{(l,k)} \rfloor > 0$, which form a fractional packing of $I_{\text{res}}^{(k)}$ and correspond to at most $d_k + 1$ bins. Since every bin is at most of size 1, we get the desired inequality. The next inequality follows from the first one and Lemma 8. The two remaining inequalities are proved by induction and the fact that $I_{\text{res}}^{(1)} \leq I^{(1)}$. □

Lemma 11. *Let k_0 be the largest value of the variable k during the execution of Algorithm Alg. Then $k_0 \leq \log_2(d_1 + 1)$ holds.*

Lemma 12. *Let J_k' be defined as above. Next-Fit packs J_k' in at most $O(\log(\frac{1}{\delta}))$ unit-sized bins.*

Lemma 13. *The following inequalities hold:*

- $LIN(I^{(k)}, C^{(1)}) - LIN(I^{(k)}_{\text{res}}, C^{(1)}) \leq LIN(I^{(k)}_{\text{int}}, C^{(1)})$,
- $LIN(I^{(k+1)}, C^{(1)}) \leq LIN(I^{(k)}_{\text{res}}, C^{(1)})$, *and*
- $LIN(I^{(k)}, C^{(1)}) - LIN(I^{(k)}_{\text{res}}, C^{(1)}) \leq LIN(I^{(k)}, C^{(1)}) - LIN(I^{(k+1)}, C^{(1)})$.

Proof (Theorem 9). Let $Bins(L)$ denote the volume of the bins into which the algorithm has packed the instance $(L, C^{(1)})$. Obviously, the solution found by Alg has the total objective value $\sum_{k=1}^{k_0}(Bins(I^{(k)}_{\text{int}}) + Bins(J_k'))$.

First, we derive a bound for $Bins(I^{(k)}_{\text{int}})$. For an instance $(I^{(k)}, C^{(1)})$ with approximate packing $v^{(k)} = (v_j^{(l,k)})$ found by Alg, we have the following inequality:

$$\begin{aligned}\sum_{l=1}^{M} c_l \sum_{j=1}^{q(\delta,d,l)} v_j^{(l,k)} &= \sum_{l=1}^{M} c_l \sum_{j=1}^{q(\delta,d,l)} \left\lfloor v_j^{(l,k)} \right\rfloor + \sum_{l=1}^{M} c_l \sum_{j=1}^{q(\delta,d,l)} \left(v_j^{(l,k)} - \left\lfloor v_j^{(l,k)} \right\rfloor \right) \\ &\leq (1+\varepsilon)\, LIN(I^{(k)}, C^{(1)}) \ .\end{aligned}$$

Since $(v_j^{(l,k)} - \lfloor v_j^{(l,k)} \rfloor)$ is a fractional packing of $(I^{(k)}_{\text{res}}, C^{(1)})$ with $LIN(I^{(k)}_{\text{res}}, C^{(1)}) \leq \sum_l c_l \sum_j (v_j^{(l,k)} - \lfloor v_j^{(l,k)} \rfloor)$, together with Lemma 13 we get

$$\begin{aligned}Bins\left(I^{(k)}_{\text{int}}\right) &= \sum_{l=1}^{M} c_l \sum_{j=1}^{q(\delta,d,l)} \left\lfloor v_j^{(l,k)} \right\rfloor \\ &\leq (1+\varepsilon)\, LIN(I^{(k)}, C^{(1)}) - \sum_{l=1}^{M} c_l \sum_{j=1}^{q(\delta,d,l)} \left(v_j^{(l,k)} - \left\lfloor v_j^{(l,k)} \right\rfloor \right) \\ &\leq (1+\varepsilon)\, LIN(I^{(k)}, C^{(1)}) - LIN(I^{(k)}_{\text{res}}, C^{(1)}) \\ &\leq (1+\varepsilon)\, LIN(I^{(k)}, C^{(1)}) - LIN(I^{(k+1)}, C^{(1)}) \ . \end{aligned} \tag{2}$$

We deduce with $LIN(I^{(k_0)}_{\text{res}}, C^{(1)}) \geq 0$ and $(I^{(3)}, C^{(1)}) \leq (I^{(2)}, C^{(1)}) \leq (I^{(1)}, C^{(1)})$

$$\begin{aligned}\sum_{k=1}^{k_0} Bins\left(I^{(k)}_{\text{int}}\right) &\leq \sum_{k=1}^{k_0-1} \left[(1+\varepsilon)\, LIN(I^{(k)}, C^{(1)}) - LIN(I^{(k+1)}, C^{(1)}) \right] \\ &\quad + (1+\varepsilon)\, LIN(I^{(k_0)}, C^{(1)}) - LIN(I^{(k_0)}_{\text{res}}, C^{(1)}) \\ &= LIN(I^{(1)}, C^{(1)}) + \varepsilon \sum_{k=1}^{k_0} LIN(I^{(k)}, C^{(1)}) - LIN(I^{(k_0)}_{\text{res}}, C^{(1)}) \\ &\leq LIN(I^{(1)}, C^{(1)}) + 3\varepsilon LIN(I^{(1)}, C^{(1)}) \\ &\quad + \varepsilon \sum_{k=4}^{k_0} LIN(I^{(k)}, C^{(1)}) \ . \end{aligned} \tag{3}$$

$LIN(I^{(k)}, C^{(1)}) \leq (2Area(I^{(k)})+1)$ holds because FF could always provide such a packing. Moreover, we have $2Area(I^{(k)}) \leq \frac{2}{2^{k-2}} Area(I^{(1)}) = \frac{1}{2^{k-3}} Area(I^{(1)})$ according to Lemma 10. Hence, the sum (3) can be bounded as follows:

$$\begin{aligned}\sum_{k=1}^{k_0} Bins\left(I_{\text{int}}^{(k)}\right) &\leq LIN(I^{(1)}, C^{(1)}) + 3\varepsilon LIN(I^{(1)}, C^{(1)}) \\ &\quad +\varepsilon \sum_{k=4}^{k_0} \left[\frac{1}{2^{k-3}} Area(I^{(1)}) + 1\right] \\ &\leq LIN(I^{(1)}, C^{(1)}) + 3\varepsilon LIN(I^{(1)}, C^{(1)}) \\ &\quad +\varepsilon LIN(I^{(1)}, C^{(1)}) + \varepsilon\,(k_0 - 3) \\ &\leq (1+4\varepsilon)\, LIN(I^{(1)}, C^{(1)}) + \varepsilon \log_2(d_1+1) \ . \end{aligned} \tag{4}$$

For the last two inequalities, we have used $Area(I^{(1)}) \leq LIN(I^{(1)}, C^{(1)})$ and $k_0 \leq \log_2(d_1+1)$ (see Lemma 11). Finally, $\sum_{k=1}^{k_0} Bins(J'_k) = O(\log(\frac{1}{\delta}) \log(d_1))$ holds because $Bins(J'_k) = O(\log(\frac{1}{\delta}))$ (see Lemma 12) and $k_0 \leq \log_2(d_1+1)$. By combining this with (4) and $LIN(I^{(1)}, C^{(1)}) \leq OPT(I^{(1)}, C^{(1)})$, we obtain the desired result. □

Remark 14. The LPs of the instances $(I^{(k)}, C^{(1)})$ are only approximately solved in contrast to Shmonin's proof [20,23] where optimal solutions are used. This results in the additional additive terms $\varepsilon LIN(I^{(k)}, C^{(1)})$ and $\varepsilon \sum_k LIN(I^{(k)}, C^{(1)})$ in (2) and (3). Thus, it is necessary to prove that the sum is bounded by $\sum_k \frac{1}{2^{k-3}} Area(I^{(k)})$, which yields the final estimate (4).

Theorem 15. *Let $\varepsilon > 0$, and let $(I^{(1)}, C^{(1)})$ be a VBP instance with d_1 item sizes, M_1 bin sizes and $s(a) \geq \delta$ for $a \in I^{(1)}$. Then Alg has a running time of*

$$\begin{aligned} O\Bigg(& d_1\left(\log(d_1) + \frac{1}{\varepsilon^2}\right) \max\left\{M_1 \cdot KP\left(d_1, \frac{\varepsilon}{6}\right), O\left(d_1 \log\log\left(d_1 \frac{1}{\varepsilon}\right)\right)\right\} \\ &+ d_1^{2.594}\left(\log(d_1) + \frac{1}{\varepsilon^2}\right) + \frac{d_1}{\delta} + n\Bigg) \ . \end{aligned} \tag{5}$$

3 The General Algorithm

A general VBP instance (I, C) has to be preprocessed to apply *Alg*. For this, let $\delta := \varepsilon^2$ and $I = I_{\text{large}} \cup I_{\text{small}} := \{a \in I \mid s(a) > \delta\} \cup \{a \in I \mid s(a) \leq \delta\}$. We now use the geometric grouping introduced by Karmarkar and Karp [16] with scaling factor $k = \lceil \frac{\varepsilon Area(I)}{\log_2(\frac{1}{\delta})+1} \rceil$ to partition I_{large} (see also the cutting-stock section in [19]). We obtain two item sets G_1 and $I_{\text{large}} \setminus G_1$. The items in G_1 can be packed in at most $\varepsilon Area(I) + O(\log_2(\frac{1}{\delta}))$ unit-sized bins with NF if the items of G_1 in $]2^{-(r+1)}, 2^{-r}]$ for $r = 0$ are packed first, then the items for $r = 1, 2, \ldots, \lfloor \log_2(\frac{1}{\delta}) \rfloor$ (see [16, Proof of Lemma 5]). Since G_1 moreover contains the largest items of I_{large}, the items in $I_{\text{large}} \setminus G_1$ can be rounded up to get $I^{(1)}$

with $I^{(1)} \leq I_{\text{large}}$. As the running time depends on the number of bin sizes (see (5)), we only use a subset of the M bin sizes in C. Set $\tilde{C} := \{c \in C | c \geq \varepsilon\}$. If $M \leq \lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \rfloor + 1$, set $C^{(1)} := \tilde{C}$. If $M > \lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \rfloor + 1$, partition $\tilde{C}$ into $\bigcup_{l=0}^{\lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \rfloor} \tilde{C}_l$ with $\tilde{C}_l := \{c | c \in](1+\varepsilon)^{-(l+1)}, (1+\varepsilon)^{-l}]\}$, and let then $C^{(1)}$ consist of the largest c in every $\tilde{C}_l$. (This construction of $C^{(1)}$ is mentioned in [22].) $I^{(1)}$ has $d_1 = O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ many different item sizes and $C^{(1)}$ at most $M_1 \leq \lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \rfloor + 1$ different bin sizes. The general algorithm is presented below.

Algorithm $P\,(I, C, \varepsilon)$

(1) Preprocess instance (I, C) as above to obtain $(G_1, C^{(1)}), (I_{\text{small}}, C^{(1)})$ and $(I^{(1)}, C^{(1)})$.
(2) Pack instance $(I^{(1)}, C^{(1)})$ with algorithm $Alg(I^{(1)}, C^{(1)}, \varepsilon)$ (see Section 2).
(3) Replace the rounded-up and packed items of $(I^{(1)}, C^{(1)})$ by their original counterparts to obtain a packing of $I_{\text{large}} \setminus G_1$.
(4) Pack the items of G_1 into unit-sized bins using Next-Fit where the items in $]2^{-(r+1)}, 2^{-r}]$ for $r = 0$ are packed first, then the items for $r = 1, 2 \ldots, \lfloor \log_2(\frac{1}{\delta}) \rfloor$.
(5) Add the small items I_{small} greedily to the existing bins using Next-Fit: open a new unit-sized bin if an item cannot be added to the existing bins.

Lemma 16 below shows that the reduced bin set $C^{(1)}$ does not increase the value of an optimal or approximate solution too much, and Lemma 17 and Th. 18 sum the results up to obtain the final approximation ratio. Note that the proofs of Lemma 16 and Th. 18 use a slight modification of [18, Pr. of Th. 1].

Lemma 16. *We have* $OPT(I^{(1)}, C^{(1)}) \leq OPT(I_{\text{large}}, C^{(1)}) \leq (1+\varepsilon) OPT(I_{\text{large}}, \tilde{C}) \leq (1+4\varepsilon) OPT(I, C) + 2$.

Lemma 17. *Let* $(I_{\text{large}}, C) = (\{a \in I | s(a) \geq \delta\}, C)$ *be defined as above. Before adding the small items in Step (5), the algorithm* P *has found an integral packing* $v = (v_j^{(l)})$ *for* (I_{large}, C) *with* $\sum_l c_l \sum_j v_j^{(l)} \leq (1+17\varepsilon) OPT(I, C) + O(\log^2(\frac{1}{\varepsilon}))$.

Theorem 18. *Let* (I, C) *be a VBP instance with* n *items. For a given* $0 < \varepsilon \leq \frac{1}{2}$, *the algorithm* P *finds a solution of the instance with objective value of at most* $\sum_l c_l \sum_j v_j^{(l)} \leq (1+17\varepsilon) OPT(I, C) + O(\log^2(\frac{1}{\varepsilon}))$.

Deriving the running time is a rather straightforward calculation. The main difficulty lies in the construction of $I^{(1)}$ and G_1, which can be obtained in time $O(\log(d_1) n) = O(\log(\frac{1}{\varepsilon}) n)$ even when I_{large} is not sorted [19, p. 298]. The running time of Step (2) is already known (see Eq. (5)), the other steps can be performed in $O(\log(\frac{1}{\varepsilon})(M + n))$. As $C^{(1)}$ contains at most $\lfloor \frac{2}{\varepsilon} \ln \frac{1}{\varepsilon} \rfloor + 1$ different bin sizes, we get the following overall running time:

$$O\Bigg(\max\left\{\min\left\{M, \left\lfloor \frac{2}{\varepsilon}\ln\frac{1}{\varepsilon}\right\rfloor + 1\right\} \cdot KP\left(d_1, \frac{\bar{\varepsilon}}{6}\right), O\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\varepsilon}\right)\log\log\left(\frac{1}{\varepsilon}\right)\right)\right\}$$
$$\cdot \frac{1}{\varepsilon}\log\left(\frac{1}{\varepsilon}\right)\cdot\left[\log\left(\frac{1}{\varepsilon}\right)+\frac{1}{\varepsilon^2}\right] + \frac{1}{\varepsilon^{2.594}}\cdot\log^{2.594}\left(\frac{1}{\varepsilon}\right)\left[\log\left(\frac{1}{\varepsilon}\right)+\frac{1}{\varepsilon^2}\right]$$
$$+\frac{1}{\varepsilon^3}\log\left(\frac{1}{\varepsilon}\right)+\log\left(\frac{1}{\varepsilon}\right)(M+n)\Bigg). \tag{6}$$

The last missing part of the algorithm is an FPTAS for the unbounded knapsack problem. Lawler's classic algorithm [17] has a running time of $KP(\tilde{n},\tilde{\varepsilon}) = O(\tilde{n} + \frac{1}{\tilde{\varepsilon}^3})$, i.e. $KP(d_1, \frac{\bar{\varepsilon}}{6}) = O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon^3})$ because of $\bar{\varepsilon} = \Theta(\varepsilon)$. The general running time can be obtained by inserting this expression into the estimates above and by simplifying them.

Theorem 19. *$P(I, C, \varepsilon)$ has for VBP a running time of $O(\frac{1}{\varepsilon^7}\log^2(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon})$ $(M + n))$.*

Remark 20. The complexity for BP is obtained from (6) with $M = 1$. Note that we can even set $\delta = O(\varepsilon)$ for BP as done e.g. by Karmarkar and Karp [16]. Moreover, the approximation ratio of $(1 + \varepsilon)$ for BP and VBP is achieved by replacing ε by $\frac{\varepsilon}{17}$, which does not change the asymptotic running time.

4 Concluding Remarks

We have found an AFPTAS for BP and VBP with improved running time and approximation ratio $(1+\varepsilon)OPT(I, C) + O(\log^2(\frac{1}{\varepsilon}))$. The algorithm is a practical application of Shmonin's theoretical result [23] combined with the algorithm by Grigoriadis et al. [12] applied to VBP. To obtain the asymptotic approximation ratio of $(1 + O(\varepsilon))$, it is shown that the number of item sizes and therefore $Area(I^{(k)})$ halves in every iteration.

Further running time improvements can probably be achieved. For instance, several unbounded knapsack problems have to be solved for column generation. At the moment, the authors consider an adapted FPTAS that takes advantage of values already calculated for smaller bins and therefore has a faster running time.

An interesting question is whether the techniques presented in this paper can be applied to other problems to find AFPTAS with improved additive terms. Bin Covering might be such a problem, for which Jansen and Solis-Oba [14] found an AFPTAS with additive term $O(\frac{1}{\varepsilon^3})$. Other examples may be Class Constrained Bin Packing [6], Bin Packing with Size Preserving Fragmentation and Bin Packing with Size Increasing Fragmentation [21,22].

References

1. Aizatulin, M., Diedrich, F., Jansen, K.: Experimental results in approximation of max-min resource sharing (unpublished manuscript)
2. Beling, P.A., Megiddo, N.: Using fast matrix multiplication to find basic solutions. Theoretical Computer Science 205(1-2), 307–316 (1998)
3. Diedrich, F.: Approximation Algorithms for Linear Programs and Geometrically Constrained Packing Problems. Ph.D. thesis, Christian-Albrechts-University to Kiel (2009)
4. Dósa, G.: The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is $FFD(I) \leq 11/9OPT(I) + 6/9$. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 1–11. Springer, Heidelberg (2007)
5. Eisemann, K.: The trim problem. Management Science 3(3), 279–284 (1957)
6. Epstein, L., Imreh, C., Levin, A.: Class constrained bin packing revisited. Theoretical Computer Science 411(34-36), 3073–3089 (2010)
7. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica 1(4), 349–355 (1981)
8. Friesen, D.K., Langston, M.A.: Variable sized bin packing. SIAM Journal on Computing 15(1), 222–230 (1986)
9. Garey, M., Johnson, D.: Computers and Intractability. A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
10. Gilmore, P., Gomory, R.: A linear programming approach to the cutting stock problem. Operations Research 9(6), 849–859 (1961)
11. Gilmore, P., Gomory, R.: A linear programming approach to the cutting stock problem—Part II. Operations Research 11(6), 863–888 (1963)
12. Grigoriadis, M.D., Khachiyan, L.G., Porkolab, L., Villavicencio, J.: Approximate max-min resource sharing for structured concave optimization. SIAM Journal on Optimization 11(4), 1081–1091 (2001)
13. Jansen, K.: Approximation Algorithms for Min-Max and Max-Min Resource Sharing Problems, and Applications. In: Bampis, E., Jansen, K., Kenyon, C. (eds.) Efficient Approximation and Online Algorithms. LNCS, vol. 3484, pp. 156–202. Springer, Heidelberg (2006)
14. Jansen, K., Solis-Oba, R.: An asymptotic fully polynomial time approximation scheme for bin covering. Theor. Comput. Sci. 306(1-3), 543–551 (2003)
15. Kantorovich, L.V.: Mathematical methods of organizing and planning production. Management Science 6(4), 366–422 (1960); significantly enlarged and translated record of a report given in 1939
16. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: 23rd Annual Symposium on Foundations of Computer Science (FOCS 1982), November 3-5, pp. 312–320. IEEE Computer Society, Chicago (1982)
17. Lawler, E.L.: Fast approximation algorithms for knapsack problems. Mathematics of Operations Research 4(4), 339–356 (1979)
18. Murgolo, F.D.: An efficient approximation scheme for variable-sized bin packing. SIAM Journal on Computing 16(1), 149–161 (1987)
19. Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. Mathematics of Operations Research 20, 257–301 (1995)
20. Sebő, A., Shmonin, G.: On the integrality gap for the bin-packing problem (unpublished manuscript)

21. Shachnai, H., Tamir, T., Yehezkely, O.: Approximation schemes for packing with item fragmentation. Theory Comput. Syst. (MST) 43(1), 81–98 (2008)
22. Shachnai, H., Yehezkely, O.: Fast Asymptotic FPTAS for Packing Fragmentable Items with Costs. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 482–493. Springer, Heidelberg (2007)
23. Shmonin, G.: Parameterised Integer Programming, Integer Cones, and Related Problems. Ph.D. thesis, Universität Paderborn (June 2007)
24. Simchi-Levi, D.: New worst-case results for the bin-packing problem. Naval Research Logistics 41(4), 579–585 (1994)

Gathering an Even Number of Robots in an Odd Ring without Global Multiplicity Detection[⋆]

Sayaka Kamei, Anissa Lamani, Fukuhito Ooshita, and Sébastien Tixeuil

[1] Dept. of Information Engineering, Hiroshima University, Japan
[2] MIS, Jules Verne University, Amiens, France
[3] Graduate School of Information Science and Technology, Osaka University, Japan
[4] Université Pierre et Marie Curie - Paris 6, LIP6-CNRS 7606, France

Abstract. We propose a gathering protocol for an *even* number of robots in a ring-shaped network that allows symmetric but not periodic configurations as initial configurations, yet uses only local weak multiplicity detection. Robots are assumed to be anonymous and oblivious, and the execution model is the non-atomic CORDA model with asynchronous fair scheduling. In our scheme, the number of robots k must be greater than 8, the number of nodes n on a network must be odd and greater than $k+3$. The running time of our protocol is $O(n^2)$ asynchronous rounds.

Keywords: Asynchronous Gathering, Local Weak Multiplicity Detection, Robots.

1 Introduction

We consider autonomous robots that are endowed with visibility sensors (but that are otherwise unable to communicate) and motion actuators. Those robots must collaborate to solve a collective task, namely *gathering*, despite being limited with respect to input from the environment, asymmetry, memory, etc. The area where robots have to gather is modeled as a graph and the gathering task requires every robot to reach a single vertex that is unknown beforehand, and to remain there hereafter.

Robots operate in *cycles* that comprise *look*, *compute*, and *move* phases. The look phase consists in taking a snapshot of the other robots positions using its visibility sensors. In the compute phase, a robot computes a target destination among its neighbors, based on the previous observation. The move phase simply consists in moving toward the computed destination using motion actuators. We consider an asynchronous computing model, *i.e.*, there may be a finite but unbounded time between any two phases of a robot's cycle. Asynchrony makes the problem hard since a robot can decide to move according to an old snapshot of the system and different robots may be in different phases of their cycles at the same time. Moreover, the robots that we consider here have weak capacities: they are *anonymous* (they execute the same protocol and have no mean to distinguish themselves from the others), *oblivious* (they have no memory that is persistent between two cycles), and have no compass whatsoever (they are unable to agree on a common direction or orientation in the ring).

[⋆] This work is supported in part by KAKENHI no.22700074.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 542–553, 2012.

While most of the literature on coordinated distributed robots considers that those robots are evolving in a *continuous* two-dimensional Euclidean space and use visual sensors with perfect accuracy that permit to locate other robots with infinite precision, a recent trend was to shift from the classical continuous model to the *discrete* model. In the discrete model, space is partitioned into a *finite* number of locations. This setting is conveniently represented by a graph, where nodes represent locations that can be sensed, and where edges represent the possibility for a robot to move from one location to the other. For each location, a robot is able to sense if the location is empty or if robots are positioned on it (instead of sensing the exact position of a robot). Also, a robot is not able to move from a position to another unless there is explicit indication to do so (*i.e.*, the two locations are connected by an edge in the representing graph). The discrete model permits to simplify many robot protocols by reasoning on finite structures (*i.e.*, graphs) rather than on infinite ones.

Related Work. In this paper, we focus on the *gathering* problem in the discrete setting where a set of robots has to gather in one single location, not defined in advance, and remain on this location [4,2,6,7,5,1]. Several deterministic algorithms have been proposed to solve the gathering problem in a ring-shaped network, which enables many problems to appear due to the high number of symmetric configurations. The case of anonymous, asynchronous and oblivious robots was investigated only recently in this context. It should be noted that if the configuration is periodic and edge symmetric, no deterministic solution can exist [7]. The first two solutions [7,6] are complementary: [7] is based on breaking the symmetry whereas [6] takes advantage of symmetries. However, both [7] and [6] make the assumption that robots are endowed with the ability to distinguish nodes that host one robot from nodes that host two robots or more in the entire network (this property is referred to in the literature as *global* weak multiplicity detection). This ability weakens the gathering problem because it is sufficient for a protocol to ensure that a single multiplicity point exists to have all robots gather in this point, so it reduces the gathering problem to the creation of a single multiplicity point. Nevertheless, the case of an even number of robots proved difficult [3,1] as more symmetric situations must be taken into account.

Investigating the feasibility of gathering with weaker multiplicity detectors was recently addressed in [4,5]. In those papers, robots are only able to test that their current hosting node is a multiplicity node (*i.e.* hosts at least two robots). This assumption (referred to in the literature as *local* weak multiplicity detection) is obviously weaker than the global weak multiplicity detection, but is also more realistic as far as sensing devices are concerned. The downside of [4] compared to [6] is that only rigid configurations (*i.e.* non symmetric configuration) are allowed as initial configurations (as in [7]), while [6] allowed symmetric but not periodic configurations to be used as initial ones. Also, [4] requires that $k < n/2$ even in the case of non-symmetric configurations, where k denotes the number of robots and n the size of the ring, respectively. By contrast, [5] proposed a gathering protocol that could cope with symmetric yet aperiodic configurations and only made use a local weak multiplicity detector, allowing k to be between 3 and $n-3$. However, [5] requires an odd number of robots, which permits to avoid a number of possibly problematic symmetric configurations.

Our Contribution. We propose a gathering protocol for an *even* number of robots in a ring-shaped network that allows symmetric but not periodic configurations as initial configurations, yet uses only local weak multiplicity detection. Robots are assumed to be anonymous and oblivious, and the execution model is the non-atomic CORDA model with asynchronous fair scheduling. For the even number of robots setting, our protocol allows the largest set of initial configurations (with respect to impossibility results) yet uses the weakest multiplicity detector to date. In our scheme, k must be greater than 8, n must be odd and greater than $k+3$. The running time of our protocol is $O(n^2)$ asynchronous rounds.

Outline. The paper is organized as follow: we first define our model in Section 2, we then present our algorithm in Section 3. Due to the lack of space, the complete proofs of correctness are omitted[1]. Finally we conclude the paper in Section 5.

2 Preliminaries

System Model. In the following, we use the same model as in [5]. We considered in this paper is similar We consider the case of an anonymous, unoriented and undirected ring of n nodes u_0,u_1,..., $u_{(n-1)}$ such that u_i is connected to both $u_{(i-1)}$ and $u_{(i+1)}$ and $u_{(n-1)}$ is connected to u_0. We assume n odd. Since no labelling is enabled (anonymous), there is no way to distinguish between nodes, or between edges.

On this ring, k robots operate in distributed way in order to accomplish a common task that is to gather in one location not known in advance. We assume that k is even. The set of robots considered here are *identical*; they execute the same program using no local parameters and one cannot distinguish them using their appearance, and are *oblivious*, which means that they have no memory of past events, they can't remember the last observations or the last steps taken before. In addition, they are unable to communicate directly, however, they have the ability to sense the environment including the position of the other robots. Based on the configuration resulting of the sensing, they decide whether to move or to stay idle. Each robot executes cycles infinitely many times, (1) first, it catches a sight of the environment to see the position of the other robots (look phase), (2) according to the observation, it determines a neighboring target destination (compute phase), (3) if it decides to move, it moves to its neighboring node towards a target destination (move phase). At instant t, a subset of robots is activated by an entity known as *the scheduler*. The scheduler can be seen as an external entity that selects some robots for the execution. This scheduler is considered to be fair *i.e,* all robots must be activated infinitely many times. The model considered in this paper is the *CORDA model* [8]. This model enables the interleaving of phases by the scheduler (For instance, one robot can perform a look operation while another is moving). In our case we add the following constraint: the Move operation is instantaneous *i.e,* when a robot takes a snapshot of its environment, it sees the other robots on nodes and not on edges. However, since the scheduler is allowed to interleave the different operations, robots can move according to an outdated snapshot since during the Compute phase, some robots may have moved.

[1] The complete proofs can be found at `http://hal.inria.fr/hal-00709074`

During the process, some robots move and occupy nodes of the ring at any time, their positions form a configuration of the system at that time. We assume that, at instant $t = 0$ (*i.e,* at the initial configuration), some of the nodes on the ring are occupied by robots, such that, each node contains at most one robot. A node u_i is said *empty* if there is no robot on u_i. The segment $[u_p, u_q]$ is defined by the sequence $(u_p, u_{p+1}, \cdots, u_{q-1}, u_q)$ of consecutive nodes in the ring, such that all the nodes of the sequence are empty except u_p and u_q that contain at least one robot. The distance D^t_p of segment $[u_p, u_q]$ in the configuration of time t is equal to the number of nodes in $[u_p, u_q]$ minus 1. We define a *hole* as the maximal set of consecutive empty nodes. That is, in the segment $[u_p, u_q]$, $(u_{p+1}, \cdots, u_{q-1})$ is a hole. The size of a hole is the number of empty nodes that compose it, the border of the hole are the two empty nodes who are part of this hole, having one robot as a neighbor.

We say that there is a *multiplicity* at some node u_i, if at this node there is more than one robot (Recall that this multiplicity is distinguishable only locally).

When a robot takes a snapshot of the current configuration on node u_i at time t, it has a *view* of the system at this node. In the configuration $C(t)$, we assume $[u_1, u_2]$,$[u_2, u_3]$,$\cdots$, $[u_w, u_1]$ are consecutive segments in a given direction of the ring. Then, the view of a robot on node u_1 at $C(t)$ is represented by
$(\max\{(D^t_1, D^t_2, \cdots, D^t_w), (D^t_w, D^t_{w-1}, \cdots, D^t_1)\}, m^t_1)$, where m^t_1 is true if there is a multiplicity at this node, and sequence $(a_i, a_{i+1}, \cdots, a_j)$ is larger than $(b_i, b_{i+1}, \cdots, b_j)$ if there is $h(i \leq h \leq j)$ such that $a_l = b_l$ for $i \leq l \leq h-1$ and $a_h > b_h$. It is stressed from the definition that robots don't make difference between a node containing one robot and those containing more. However, they can detect m^t of the current node, *i.e.* whether they are alone on the node or not (they have a local weak multiplicity detection).

Configurations that have no multiplicity are classified into three classes in [7]. A configuration is said to be *periodic* if it is represented by a configuration of at least two copies of a sub-sequence. A configuration is said to be *symmetric* if the ring contains a single axis of symmetry. Otherwise, the configuration is said to be *rigid*. For these configurations, the following lemma is proved in [7].

Lemma 1. *If a configuration is rigid, all robots have distinct views. If a configuration is symmetric and non-periodic, there exists exactly one axis of symmetry.*

This implies that, if a configuration is symmetric and non-periodic, at most two robots have the same view.

We now define some useful terms that will be used to describe our algorithm. We denote by the *inter-distance d* the minimum distance taken among distances between each pair of distinct robots (in term of the number of edges). Given a configuration of inter-distance d, a *d.block* is any maximal elementary path where there is one robot every d edges. The border of a d.block consists in the two external robots of the d.block. The size of a d.block is the number of robots that it contains. We refer to the d.block which has the biggest size by *biggest d.block*. A robot that is not in any d.block is said to be an *isolated robot*.

We evaluate the time complexity of our algorithm with asynchronous rounds. An asynchronous round is defined as the shortest fragment of an execution where each robot performs a move phase at least once.

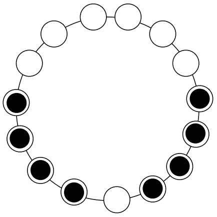


Fig. 1. Terminal Configuration

Problem to Be Solved. The problem considered in our work is the gathering problem, where starting from any arbitrary, non periodic configuration without multiplicities, k robots have to gather in one location not known in advance before stopping there forever.

3 Proposed Algorithm

We propose in this section an algorithm that solves the gathering problem starting from any non-periodic configuration on a ring provided that n is odd, k is even, $k > 8$ and $n > k + 3$.

In the following, a configuration is said to be *Terminal* if it is symmetric and contains two 1.blocks of size $k/2$ at distance 2 from each other (refer to Figure 1).

The algorithm comprises three main phases as follow:

- **Phase 1.** The aim of this phase is to reach a configuration with either a single 1.block of size k or two 1.blocks of the same size $k/2$. The initial configuration of this phase is any arbitrary non-periodic configuration without multiplicities.
- **Phase 2.** The goal of this phase is to reach *Terminal* configuration (refer to Figure 1) starting from any configuration with either a single 1.block or two 1.blocks of the same size. In the following C_{sp} includes any configuration that can occur during Phase 2.
- **Phase 3.** The starting configuration of this phase is the *Terminal* configuration. During this phase, robots perform the gathering such that at the end of this phase, all robots are on the same node.

During the execution of Phase 1, if a configuration part of C_{sp} is reached, the algorithm immediately moves to Phase 2.

When the configuration is symmetric, and since the number of nodes is odd, the axis of symmetry intersects with the ring on exactly one node and one edge. Let us refer to such a node by S. Additionally, because the number of robots is even, there is no robot on S. Observe that the size of the hole including S is odd. Let us call such a hole the *Leader hole* and let us refer to it by H. The other hole on the axis of symmetry is called the *Slave hole*. Both the *Leader Hole* and the *Slave Hole* can be part of a d.block.

3.1 Phase 1 Algorithm

Starting from a non periodic configuration without multiplicities, the aim of this phase is to reach a configuration with either a single 1.block of size k or two 1.blocks of size $k/2$.

The idea of this phase is the following: In the initial configuration, in the case where the configuration contains isolated robots, these isolated robots are the ones that can move in order to join a d.block. Thus, in a finite time, the configuration contains only d.blocks. If all the d.blocks have the same size, robots move such that there will be at least two d.blocks in the configuration that have different size. Robots then move towards the closest biggest d.block. In order to avoid creating periodic configurations, only some robots are allowed to move, these robots are the ones that have the biggest view. Depending on the nature of the configuration (symmetric or rigid), only one d.block (respectively two d.blocks if the configuration is symmetric) becomes the biggest d.block in the configuration (let refer to the set of these d.block by target blocks). These d.blocks are then the target of all the other robots that move in order to join them. When all the robots are in a d.blocks part of the set target blocks then in the case $d > 1$, some robots move in order to decrease d. After that, the system will have the same behavior unless $d = 1$.

The following configurations are considered:

- ***BlockDistance*** Configuration. In this configuration, there are only two d.blocks of the same size ($k/2$) or a single d.block of size k such that in both cases $d > 1$. Note that the configuration is symmetric and does not contain any isolated robot. The robots allowed to move are the ones that are neighbors of H. Their destination is their adjacent empty node in the opposite direction of H.

- ***BlockMirror*** Configuration. In such a configuration there are only d.blocks of the same size and no isolated robots. The number of d.blocks is bigger than 2. Two sub configurations are possible as follow:

    - ***BlockMirror1*** Configuration. The configuration is in this case rigid. The robot allowed to move is the one that is the closest to a d.block. If there are two or more such robots, then the one having the biggest view among them is allowed to move. Its destination is its adjacent empty node towards the closest neighboring d.block.

    - ***BlockMirror2*** Configuration. The configuration is in this case symmetric. Let the d.blocks that are neighbors to H or including H be the guide blocks. The robots allowed to move are the ones that share a hole other than H with the guide blocks. Their destinations are their adjacent empty node towards the closest guide block.

- ***BigBlock*** Configuration. In this configuration, the configuration is neither *BlockMirror* nor *BlockDistance*. Then, there is at least one d.block that has the biggest size. Two sub configurations are defined as follow:

- ***BigBlock1*** Configuration. In this case there is at least one isolated robot that shares a hole with one of the biggest d.blocks. Two sub-cases are possible as follow:

  * ***BigBlock1-1*** Configuration. The configuration is rigid and contains either (i) two 1.blocks of the same size $(k-2)/2$ and two isolated robots that share a hole together or (ii) one 1.block of size $(k-2)$ and two isolated robots that share a hole together. The robot that is allowed to move in both cases is the one that is the farthest to the neighboring 1.block. Its destination is its adjacent empty node towards the neighboring 1.block.

  * ***BigBlock1-2*** Configuration. This configuration is different from *BigBlock1-1* and includes all the other configurations of *BigBlock1*. The robots allowed to move are part of the set of robots that share a hole with a biggest d.block such that they are the closest ones to a biggest d.block. If there exists more than one such robot, then only robots with the biggest view among them are allowed to move. Their destination is their adjacent empty node towards one of the nearest neighboring biggest d.blocks.

- ***BigBlock2*** Configuration. In this case there is no isolated robot that is neighbor of a biggest d.block. The robots allowed to move are the ones that are the closest to a biggest d.block. If there exist more than one such robots, then only robots with the biggest view among them can move. Their destination is their adjacent empty node towards one of the nearest neighboring biggest d.blocks.

3.2 Phase 2 Algorithm

Recall that the aim of Phase 2 is to reach *Terminal* configuration. The starting configuration of this phase is one of the configurations part of C_{sp}. When the configuration is symmetric, the two 1.blocks that are neighbors of the Leader hole (H) (respectively Slave hole) are called *the Leader block* (respectively *the Slave block*). To simplify the explanation we assume in the following that an isolated robot is also a single 1.block of size 1. The idea of this phase is to make the robots move towards the *Leader Hole* while keeping both the symmetry of the configuration and the inter-distance equal to 1. The symmetry of the configuration is maintained in the following matter: If the scheduler activates only one robot in the configuration then in the next step the robot that was supposed to move is the only one that can move. Note that this robot can be easily determined since we have an odd number of nodes and an even number of robots in the ring. Thus the robots keep moving in the same direction, towards the *Guide Hole* whose size decreases at each time. Hence, *Terminal* configuration is reached in a finite time. The algorithm of Phase 3 can then be executed.

In the following, we define the nine configurations that are part of the set C_{sp}: *Start*, *Even-T*, *Split-S*, *Split-A*, *Odd-T*, *Block*, *Biblock*, *TriBlock-S* and *TriBlock-A*. The behavior of robots in each configuration is also provided.

1. ***Start*** Configuration. This configuration is symmetric and contains two 1.blocks with size $k/2$ but not being at distance 2. The robots allowed to move are the two

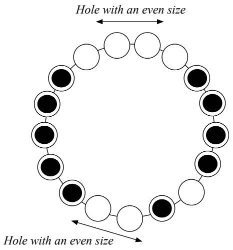


Fig. 2. $Even-T$ Configuration

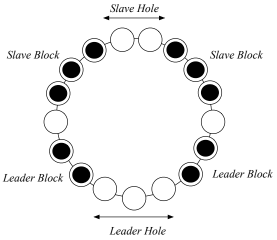


Fig. 3. $Split-S$ Configuration

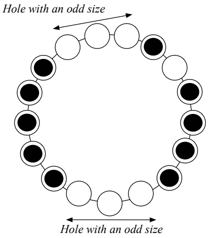


Fig. 4. $Odd-T$ Configuration

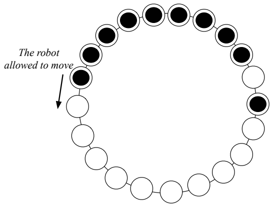


Fig. 5. $Biblock$ Configuration

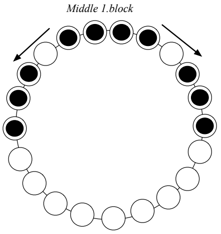


Fig. 6. $TriBlock-S$ Configuration

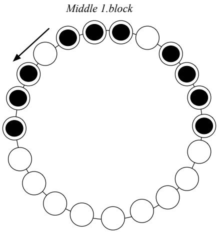


Fig. 7. $TriBlock-A$ Configuration

robots that are at the border of the 1.blocks neighboring to the Leader hole. Their destination is their adjacent empty node in the opposite direction of the 1.block they belong to.

2. ***Even-T*** Configuration. The configuration is in this case rigid and contains three 1.blocks of size respectively $k/2$, $(k/2)-1$ and 1. Additionally, the 1.block of size 1 is at distance 2 from the 1.block of size $k/2-1$. The number of holes between the 1.block of size 1 and the one of size $k/2$ is even. Note that this is also the case for the hole between the 1.block of size $(k/2)-1$ and the one of size $k/2$ (refer to Figure 2). The only robot allowed to move is the one that is at the border of the 1.block of size $k/2$ sharing a hole with the 1.block of size 1. Its destination is its adjacent empty node in the opposite direction of the 1.block it belongs to.

3. ***Split-S*** Configuration. The configuration is symmetric and contains four 1.blocks such that the 1.blocks on the same side of the axis of symmetry are at distance 2 (refer to Figure 3). The robots allowed to move in this case are the ones that are at the border of the Slave block sharing a hole with the Leader block. Their destination is their adjacent empty node towards the Leader block.

4. ***Split-A*** Configuration. The configuration is rigid and contains four 1.blocks and exactly one hole of an even size. Let the 1.blocks that are neighbors of the hole of an even size be $S1$ and $S2$, and let the other two 1.blocks be $L1$ and $L2$. $S1$ and $L1$ (respectively $S2$ and $L2$) are at distance 2 from each other. Then, $|S1| = |S2| + 1$, $|L2| = |L1| + 1$, $|S1| + |L1| = |S2| + |L2| = k/2$. The size of the hole between $L1$ and $L2$ is odd, and the distances between $L1$ and $S1$ (respectively between $L2$ and $S2$) is equal to 2. In this case, only one robot is allowed to move. This robot is the one at the border of $S1$ sharing a hole with $L1$. Its destination its adjacent empty node towards $L1$.

5. ***Odd-T*** Configuration. The configuration is rigid and contains three 1.blocks of size respectively $k/2$, $(k/2) - 1$ and 1. Additionally, the 1.block of size 1 is at distance 2 from the 1.block of size $k/2 - 1$. Observe that this configuration is different from *Even-T* Configuration since all the holes in the ring have an odd size (refer to Figure 4). The only robot allowed to move is the one that is part of the 1.block of size 1. Its destination is its adjacent empty node towards the 1.block of size $(k/2) - 1$.

6. ***Block*** Configuration. The configuration contains in this case, a single 1.block of size k. Note that the configuration is symmetric. The robots allowed to move are the ones that are at the border of the 1.block. Their destination is their adjacent node in the opposite direction of the 1.block they belong to.

7. ***Biblock*** Configuration. This configuration is rigid and contains two 1.blocks B_1 and B_2 at distance 2 from each other such that $|B_1| = k - 1$ and $|B_2| = 1$ (refer to Figure 5). The robot allowed to move is the one that is at the border of the biggest 1.block not having a neighboring occupied node at distance 2. Its destination is its adjacent node in the opposite direction of the 1.block it belongs to.

8. ***TriBlock-S*** Configuration. This configuration is symmetric and contains three 1. blocks separated by one empty node (refer to Figure 6). The robots allowed to move are the ones that are at the border of the 1.block on the axis of symmetry. Their destination is their adjacent empty node in the opposite direction of the 1.block they belong to.

9. ***TriBlock-A*** Configuration. This configuration is rigid and contains three 1.blocks (B_1, B_2 and B_3) such that there is one 1.block that is at distance 2 from both other 1.blocks (let B_1 be this 1.block, refer to Figure 7). $|B_2| = |B_3| + 1$. The robot allowed to move is the one that is at the border of B_1 and the closest to B_3. Its destination is its adjacent empty node in the opposite direction of the 1.block it belongs to.

The transitions between the configurations of Phase 2 are illustrated in Figure 8.

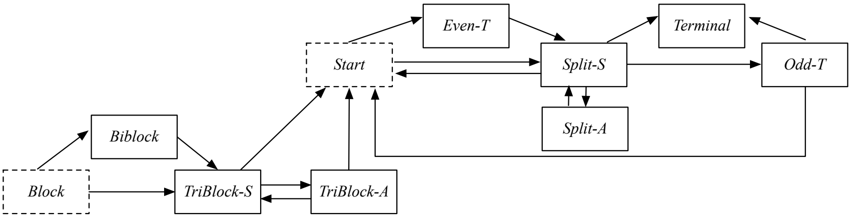


Fig. 8. Transition of the configuration by Phase 2

3.3 Phase 3 Algorithm

During this phase, robots perform the gathering such that at the end, all robots are on the same location. The starting configuration of this phase is the *Terminal* configuration.

When the *Terminal* configuration is reached at the end of the second phase (or when the configuration is built in the first phase), the only robots that can move are the ones that are at the extremity of a 1.block being neighbors of a hole of size 1. Since the configuration is symmetric there are exactly two robots allowed to move. Two cases are possible as follow:

1. The scheduler activates both robots at the same time. In this case the configuration remains symmetric and a multiplicity is created on the axis of symmetry.
2. The scheduler activates only one robot. In this case the configuration that is reached becomes asymmetric and contains two 1.blocks B_1 and B_2 at distance 2 such that $|B_2| = |B_1| - 2$ (one robot from B_2 has moved and joined B_1, let this robot be r_1). Note that this configuration is easily recognizable by robots. The robot that is in B_1 being neighbor of r_1 is the one allowed to move. Its destination is its adjacent occupy node towards r_1. Note that once it moves, the configuration becomes symmetric and a multiplicity is created on the node that is on the axis of symmetry. Let us refer to such symmetric configuration with a multiplicity *Target* configuration.

In the *Target* configuration, robots that are part of the multiplicity are not allowed to move anymore. For the other robots, they can only see an odd number of robots in the configuration since they are enable to see the multiplicity on the axis of symmetry (recall that they have a local week multiplicity detection). In addition, since they are oblivious, they cannot remember their number before reaching *Target* configuration. On the other hand, an algorithm has been proposed in [5] that solves the gathering problem from such a configuration (Phase 2 in [5]). Robots can then execute the same algorithm to perform the gathering.

4 Proof of Correctness

In the following, we prove the correctness of our algorithm. We define an *outdated robot* as the robot that takes a snapshot at instant t but moves only at instant $t + j$ where

$j > 0$. Thus, when such a robot moves, it does so based on an outdated perception of the configuration. Additionally, we define an *outdated robot with an incorrect target* as the outdated robot that its target destination is incorrect in the current configuration, i.e., if the robot takes a new snapshot to the current configuration, the computed destination is different.

- **Phase 1**

  To prove the correctness of Phase 1 algorithm, we first show that there exists in the configurations at most one outdated robot with an incorrect target. Next, we prove that starting from non periodic configuration without any multiplicities, neither a periodic configuration nor a configuration with a multiplicity is reached during the execution of Phase 1. Finally, we show that either *Terminal* configuration or a configuration $C^* \in C_{sp}$ is eventually reached. Thus:

  Theorem 1. *From any non-periodic initial configuration C without any multiplicities, either Terminal configuration or a configuration $C^* \in C_{sp}$ is reached in $O(n^2)$ rounds.*

- **Phase 2**

  To prove the correctness of Phase 2 algorithm, we first show that when a configuration $C^* \in C_{sp}$ is reached, there exist no outdated robots with incorrect target. We then show that *Terminal* configuration is eventually reached. Hence:

  Theorem 2. *Starting from any configuration $C^* \in C_{sp}$, Terminal configuration is reached in $O(kn)$ rounds.*

- **Phase 3**

  The correctness of Phase 3 algorithm is proven by showing first that when *Terminal* configuration is reached, it does not contain any outdated robots with incorrect target. *Target* configuration is then proven to be eventually reached. Proofs in [5] are then used to show that the gathering is performed in $O(k^2)$ rounds.

The following Theorem holds:

Theorem 3. *Starting from any non-periodic initial configuration without any multiplicities, the gathering is performed in $O(n^2)$ rounds.*

5 Conclusion

We presented a gathering protocol for an even number of anonymous and oblivious robots that are initially located on different nodes of a ring, and are endowed with a weak local multiplicity detector only. Our gathering can start from any configuration that is not periodic, yet expects the ring to have an odd size. This constraint permits to avoid edge-edge symmetries in the initial configurations, as they are known to be ungatherable [7].

If we relax the constraint on the parity of the ring size (that is, the ring is even) but maintain the absence of edge-edge symmetry and periodicity requirement (that are mandatory for problem solvability), no node-edge symmetry can actually occur in the initial configuration. If the initial configuration is rigid, it is known that gathering with local weak multiplicity detection is feasible [4]. There remains the case of the initial node-node symmetry. With *global* weak multiplicity detection, this case is solvable for 6 robots [1] or more than 18 robots [7]. A similar characterization using only *local* weak multiplicity detection looks challenging.

References

1. D'Angelo, G., Di Stefano, G., Navarra, A.: Gathering of Six Robots on Anonymous Symmetric Rings. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 174–185. Springer, Heidelberg (2011)
2. Flocchini, P., An, H.-C., Krizanc, D., Santoro, N., Sawchuk, C.: Multiple Mobile Agent Rendezvous in a Ring. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 599–608. Springer, Heidelberg (2004)
3. Haba, K., Izumi, T., Katayama, Y., Inuzuka, N., Wada, K.: On gathering problem in a ring for 2n autonomous mobile robots. In: Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems, Poster (2008)
4. Izumi, T., Izumi, T., Kamei, S., Ooshita, F.: Mobile Robots Gathering Algorithm with Local Weak Multiplicity in Rings. In: Patt-Shamir, B., Ekim, T. (eds.) SIROCCO 2010. LNCS, vol. 6058, pp. 101–113. Springer, Heidelberg (2010)
5. Kamei, S., Lamani, A., Ooshita, F., Tixeuil, S.: Asynchronous Mobile Robot Gathering from Symmetric Configurations without Global Multiplicity Detection. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 150–161. Springer, Heidelberg (2011)
6. Klasing, R., Kosowski, A., Navarra, A.: Taking advantage of symmetries: Gathering of many asynchronous oblivious robots on a ring. Theoretical Computer Science 411(34-36), 3235–3246 (2010)
7. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theoretical Computer Science 390(1), 27–39 (2008)
8. Prencipe, G.: CORDA: Distributed coordination of a set of autonomous mobile robots. In: Proc. 4th European Research Seminar on Advances in Distributed Systems (ERSADS), pp. 185–190 (2001)

Reversal Hierarchies for Small 2DFAs

Christos A. Kapoutsis[1,*] and Giovanni Pighizzini[2]

[1] LIAFA, Université Paris VII, France
[2] DI, Università degli Studi di Milano, Italia

Abstract. A two-way deterministic finite automaton *with* $r(n)$ *reversals* performs $\leq r(n)$ input head reversals on every n-long input. Let $2\mathsf{D}[r(n)]$ be all families of problems solvable by such automata of size polynomial in the index of the family. Then the *reversal hierarchy* $2\mathsf{D}[0] \subseteq 2\mathsf{D}[1] \subseteq 2\mathsf{D}[2] \subseteq \cdots$ is strict, but $2\mathsf{D}[O(1)] = 2\mathsf{D}[o(n)]$. Moreover, the *inner-reversal hierarchy* $2\mathsf{D}(0) \subseteq 2\mathsf{D}(1) \subseteq 2\mathsf{D}(2) \subseteq \cdots$, where now the bound is only for reversals strictly between the input end-markers, is also strict.

1 Introduction

A long-standing open question of the Theory of Computation is whether every two-way nondeterministic finite automaton (2NFA) is equivalent to a deterministic one (2DFA) with only polynomially more states; or, in other terms, whether $2\mathsf{D} = 2\mathsf{N}$, where $2\mathsf{D}$ and $2\mathsf{N}$ are the classes of (families of) problems which are solvable by 'small' (i.e., polynomial-size) 2DFAs and 2NFAs, respectively [8].

In 2002, J. Hromkovič suggested approaching this question by its variants for 2DFAs with restricted number of input-head reversals [3]. Specifically, let a '2DFA *with* $r(n)$ *reversals*' be one which performs $\leq r(n)$ reversals on every n-long input. Next, for any class $\mathcal{R}$ of natural functions, let $2\mathsf{D}[\mathcal{R}]$ be the restriction of $2\mathsf{D}$ to problems that are solvable by small 2DFAs with $r(n)$ reversals, for some $r \in \mathcal{R}$. Then, the following obvious inclusions hold, where $\mathsf{0}, \mathsf{1}, \ldots$ are singletons for the individual constant functions, and const is all these functions together:

$$\begin{aligned} &2\mathsf{D}[\mathsf{0}] \underset{\text{(a)}}{\subseteq} 2\mathsf{D}[\mathsf{1}] \underset{\text{(b)}}{\subseteq} 2\mathsf{D}[\mathsf{2}] \subseteq \cdots \subseteq 2\mathsf{D}[\mathsf{r}] \subseteq \cdots \\ &\qquad \subseteq 2\mathsf{D}[\mathsf{const}] \subseteq 2\mathsf{D}[O(1)] \subseteq 2\mathsf{D}[o(n)] \overset{\text{(c)}}{\subseteq} 2\mathsf{D}[O(n)] \overset{\text{(d)}}{\subseteq} 2\mathsf{D}\,. \end{aligned} \tag{1}$$

Hromkovič suggested resolving all these inclusions, as well as every relationship between a class and its counterpart for 2NFAs [3, Research Problems 2–4].

Some answers are known: (a), (b), (c) are strict, by [7, Prop. 1], [1, Th. 2.2], and [4, Th. 3]; (d) is equality, as small 2DFAs have small halting equivalents [9,2], which reverse $O(n)$ times [4, Fact 3]; and every class up to $2\mathsf{D}[o(n)]$ is strictly inside its counterpart for 2NFAs, as the witness to [4, Th. 1] admits small 2NFAs even with 0 reversals. Here, we resolve most of the remaining inclusions in (1).

We start in Sect. 3, with a crossing-sequence argument which proves that a 2DFA cannot reverse $o(n)$ times unless it already reverses $O(1)$ times (and thus the lower bound of [4, Th. 1] is really a bound for 2DFAs with $O(1)$ reversals).

[*] Supported by a Marie Curie Intra-European Fellowship (PIEF-GA-2009-253368) within the European Union Seventh Framework Programme (FP7/2007-2013).

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 554–565, 2012.

Theorem 1. *Every* 2DFA *with $o(n)$ reversals is a* 2DFA *with $O(1)$ reversals.*

We continue in Sect. 4, with a uniform argument for all $r \geq 1$, which proves that small 2DFAs with r reversals are strictly more powerful than small 2DFAs with $< r$ reversals. Crucially, our argument makes black-box use of [4, Th. 2].

Theorem 2. *Let $r \geq 1$. For each $h \geq 1$, some problem requires $2^{\Omega(h/r)}$ states on every* 2DFA *with $< r$ reversals, but only $O(r+h)$ states on a* 2DFA *with r reversals.*

Hence, with Theorems 1 and 2 counted in, the chain of (1) is updated as follows:

$$\begin{aligned} &\mathsf{2D}[0] \underset{[7]}{\overset{*}{\subsetneq}} \mathsf{2D}[1] \underset{[1]}{\overset{*}{\subsetneq}} \mathsf{2D}[2] \overset{*}{\subsetneq} \cdots \overset{*}{\subsetneq} \mathsf{2D}[\mathsf{r}] \overset{*}{\subsetneq} \cdots \\ &\qquad\qquad\qquad \underset{*}{\subsetneq} \mathsf{2D}[\mathsf{const}] \underset{?}{=} \mathsf{2D}[O(1)] \underset{*}{=} \mathsf{2D}[o(n)] \overset{[4]}{\subsetneq} \mathsf{2D}[O(n)] \overset{[9,2]}{=} \mathsf{2D}\,, \end{aligned} \tag{2}$$

where '$*$' marks our contributions, and '?' marks the only remaining unresolved inclusion —we conjecture that the seemingly obvious equality is indeed true.

Finally, in Sect. 5 we show that Theorem 2 remains valid even when we bound only the *inner reversals*, which occur strictly between the two input end-markers (as opposed to *outer reversals*, which occur on the end-markers). Crucially, our proof builds on a stronger variant of the argument behind [4, Th. 2].

Theorem 3. *Let $r \geq 1$. For each $h \geq 1$, some problem requires $\Omega(2^{h/2})$ states on every* 2DFA *with $< r$ inner reversals, but only $O(h)$ states on a* 2DFA *with r inner reversals (and 0 outer reversals, if r is even; or 1 outer reversal, if r is odd).*

Thus, an additional *inner-reversal hierarchy* $\mathsf{2D}(0) \subsetneq \mathsf{2D}(1) \subsetneq \cdots \subsetneq \mathsf{2D}(\mathsf{const})$ is established, where now $\mathsf{2D}(\mathcal{R})$ restricts $\mathsf{2D}$ to problems solvable by small 2DFAs with $r(n)$ *inner* reversals, for some $r \in \mathcal{R}$. (Clearly, $\mathsf{2D}[\mathcal{R}] \subseteq \mathsf{2D}(\mathcal{R})$ for all $\mathcal{R}$; moreover, from const upwards this inclusion is easily seen to be an equality.)

2 Preparation

If $h \geq 0$, then $[h] := \{0, \ldots, h-1\}$. If S is a set, then $|S|$, $\overline{S}$, $\mathbb{P}(S)$, $S_\perp$ are its size, complement, powerset, and augmentation $S \cup \{\perp\}$. If f, g are partial functions, then the composition $(f \circ g)(a)$ is defined iff both $f(a)$ and $g(f(a))$ are, and then equals $g(f(a))$; the k-fold composition of f with itself is denoted by f^k.

Let Σ be an alphabet. If $z \in \Sigma^*$ is a string, we write $|z|$, z_j, z^j, and z^{R} for its length, j-th symbol ($1 \leq j \leq |z|$), j-fold concatenation with itself ($j \geq 0$), and reverse; its j-th boundary ($1 \leq j \leq |z|+1$) is the left boundary of z_j, or the right one if $j = |z|+1$. If $Z \subseteq \Sigma^*$, then $Z^{\mathsf{R}} := \{z^{\mathsf{R}} \mid z \in Z\}$.

A (*promise*) *problem* over Σ is a pair $\mathfrak{L} = (L, \tilde{L})$ of disjoint subsets of Σ^*. Every w in the *promise* $L \cup \tilde{L}$ is an *instance* of $\mathfrak{L}$: *positive* if $w \in L$, or *negative* if $w \in \tilde{L}$. To solve $\mathfrak{L}$ is to accept every $w \in L$ but no $w \in \tilde{L}$.

2.1 Two-Way Automata

A *two-way deterministic finite automaton* (2DFA) is any $M = (Q, \Sigma, \delta, q_{\mathrm{s}}, q_{\mathrm{a}}, q_{\mathrm{r}})$, where Q is a set of *states*, Σ is an *alphabet*, $q_{\mathrm{s}}, q_{\mathrm{a}}, q_{\mathrm{r}} \in Q$ are the *start*, *accept*, and *reject* states, and $\delta : Q \times (\Sigma \cup \{\vdash,\dashv\}) \to Q \times \{\text{L},\text{R}\}$ is the (total) *transition function*, using two end-markers $\vdash,\dashv \notin \Sigma$ and the two directions L,R. An input $w \in \Sigma^*$ is presented to M between the end-markers, as $\vdash w \dashv$. The computation starts at q_s and on $\vdash$. At each step, the next state and head motion are derived from δ and the current state and symbol. End-markers may be violated only if the next state is q_a or q_r: $\delta(\cdot\,,\vdash)$ is always (q_a,L), (q_r,L), or $(\cdot\,,\text{R})$; and $\delta(\cdot\,,\dashv)$ is always (q_a,R), (q_r,R), or $(\cdot\,,\text{L})$. So, the computation loops, or falls off $\vdash w \dashv$ into q_r, or falls off $\vdash w \dashv$ into q_a. In this last case, we say M *accepts* w.

Formally, *the computation of M from state p and the j-th symbol of string z*, denoted $\text{COMP}_{M,p,j}(z)$, is the longest sequence $c = ((q_t, j_t))_{0 \leq t < m}$ such that $0 < m \leq \infty$, $(q_0, j_0) = (p, j)$, and every next (q_t, j_t) follows from the previous one via δ and z in the usual way (Fig. 1a). We call (q_t, j_t) the *t-th point* of c. If $m = \infty$ then c *loops*; otherwise it *halts*, and *hits left* (if $j_{m-1} = 0$) or *hits right* (if $j_{m-1} = |z|+1$) *into* q_{m-1}. The computation $\text{LCOMP}_{M,p}(z) := \text{COMP}_{M,p,1}(z)$ is *the* L*-computation of M from p on z*; depending on whether it loops, hits left, or hits right, we call it a L*-loop*, L*-turn*, or LR*-traversal*. Symmetrically, *the* R*-computation of M from p on z*, $\text{RCOMP}_{M,p}(z) := \text{COMP}_{M,p,|z|}(z)$, is a R*-loop*, R*-turn*, or RL*-traversal*. The *(full) computation of M on $w \in \Sigma^*$* is $\text{COMP}_M(w) := \text{LCOMP}_{M,q_\text{s}}(\vdash w \dashv)$. So, M accepts w iff $\text{COMP}_M(w)$ falls off $\vdash w \dashv$ into q_a.

The *j-th crossing sequence* of a computation c on a string z is the sequence $q_1, q_2, \ldots$ where q_i is the state immediately after c crosses the j-th boundary of z for the i-th time. Easily, if c halts, then every crossing sequence contains $\leq 2|Q|$ states, and thus $\leq (|Q|+1)^{2|Q|}$ of these sequences are distinct.

A *reversal* of c is a point $(\,.\,, j_t)$ whose predecessor and successor exist and lie on the same side relative to it: $t \neq 0, m-1$, and $j_{t-1}, j_{t+1} < j_t$ or $j_t < j_{t-1}, j_{t+1}$ (Fig. 1a). If c is full, a reversal is either an *outer reversal*, if it lies on $\vdash$ or $\dashv$, or an *inner reversal*, otherwise. We write $r(c)$ for the total number of reversals in c. Clearly $0 \leq r(c) \leq \infty$, with $r(c) = \infty$ iff c loops. We write $r_M(n)$ for the maximum $r(c)$ over all full computations c of M on n-long inputs. Easily, if finite, $r_M(n)$ is at most linear: $r_M(n) = \infty$ or $r_M(n) \leq |Q|\cdot(n+2)$, for all n.

We say M is a 2DFA *with $r(n)$ reversals* if $r_M(n) \leq r(n)$ for all n; or a 2DFA *with $r(n)$ inner reversals* if every full computation on an n-long input performs

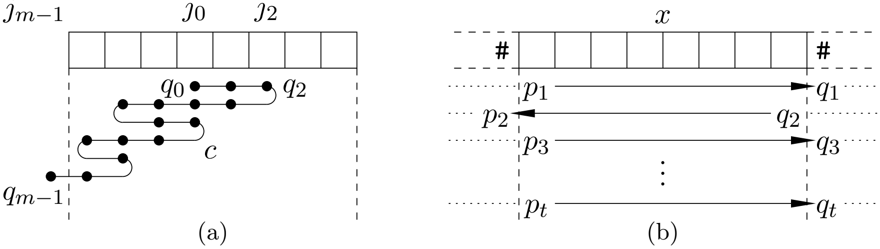


Fig. 1. (a) A left-hitting computation c with $m = 15$ and $r(c) = 5$ reversals, at points 2, 6, 8, 11, and 12. (b) A certificate for x (for the case of odd t).

$\leq r(n)$ inner reversals. If M is a 2DFA with 0 inner reversals, we call it *sweeping* (SDFA). If it is a 2DFA with 0 reversals and may also *hang* (i.e., δ is a partial function), we call it *one-way* (1DFA); then, the state q_{i} for which $\delta(q_{\mathrm{s}}, \vdash) = (q_{\mathrm{i}}, \mathrm{R})$ is called *initial*, every state q with $\delta(q, \dashv) = (q_{\mathrm{a}}, \mathrm{R})$ is called *final*, and M accepts w iff $\mathrm{LCOMP}_{M,q_{\mathrm{i}}}(w)$ hits right into a final state.

2.2 Parallel Automata

A (*two-sided*) *parallel automaton* ($\mathrm{P}_2$1DFA) [10] is any triple $M = (\mathcal{A}, \mathcal{B}, F)$ where $\mathcal{A}, \mathcal{B}$ are disjoint families of 1DFAs over an alphabet Σ, and F is a subset of the cartesian product of all sets $Q^D_\perp$, for $D \in \mathcal{A} \cup \mathcal{B}$ and Q^D the state set of D. To run M on input $w \in \Sigma^*$ means to run each D on w (without end-markers) from its initial state and record the *result* (the state in which D falls off w, or $\perp$ if it hangs), but with a twist: each $D \in \mathcal{A}$ reads w from left to right (as usual), while each $D \in \mathcal{B}$ reads w from right to left (i.e., it reads w^{R}). We say M accepts w iff the produced tuple of $|\mathcal{A}|+|\mathcal{B}|$ results is in F.

If F consists of the tuples where every result is a final state in the respective 1DFA, then M is a *parallel intersection automaton* ($\cap_2$1DFA) [8]: it accepts iff *all* its components do. If F consists of the tuples where at least one of the results is a final state, then M is a *parallel union automaton* ($\cup_2$1DFA) [8]: it accepts iff *any* of its component does. In both cases, we write only $M = (\mathcal{A}, \mathcal{B})$. When $\mathcal{B} = \emptyset$ or $\mathcal{A} = \emptyset$, we say M is *left-sided* ($\cap_{\mathrm{L}}$1DFA, $\cup_{\mathrm{L}}$1DFA) or *right-sided* ($\cap_{\mathrm{R}}$1DFA, $\cup_{\mathrm{R}}$1DFA).

We now recall notions and facts leading to *generic strings* and *blocks* [10,4].

For $D \in \mathcal{A}$ and $y \in \Sigma^*$, the set of states that can be produced on the right boundary of y by L-computations of D is denoted by:

$$Q^D_{\mathrm{LR}}(y) := \{q \mid (\exists p)[\,\mathrm{LCOMP}_{D,p}(y) \text{ hits right into } q\,]\}\,.$$

For every right extension yz of y, we let $\alpha^D_{y,z} : Q^D_{\mathrm{LR}}(y) \rightharpoonup Q^D$ be the partial function whose value on each $q \in Q^D_{\mathrm{LR}}(y)$ is either the state which $\mathrm{LCOMP}_{D,q}(z)$ hits right into, or undefined if $\mathrm{LCOMP}_{D,q}(z)$ hangs. Similarly, for $D \in \mathcal{B}$, we let $Q^D_{\mathrm{RL}}(y) := \{q \mid (\exists p)[\,\mathrm{RCOMP}_{D,p}(y) \text{ hits left into } q\,]\}$, and $\beta^D_{z,y} : Q^D_{\mathrm{RL}}(y) \rightharpoonup Q^D$ be such that $\beta^D_{z,y}(q) = r$ iff $\mathrm{RCOMP}_{D,q}(z)$ hits left into r. The next straightforward fact is a summary of [6, Facts 3.7–9] (as well as a special case of [5, Facts 3–4]).

Fact 1. *If $D \in \mathcal{A}$ then $\alpha^D_{y,z}$ partially surjects $Q^D_{\mathrm{LR}}(y)$ to $Q^D_{\mathrm{LR}}(yz)$, thus $|Q^D_{\mathrm{LR}}(y)| \geq |Q^D_{\mathrm{LR}}(yz)|$; in addition, $Q^D_{\mathrm{LR}}(yz) \subseteq Q^D_{\mathrm{LR}}(z)$. If $D \in \mathcal{B}$ then $\beta^D_{z,y}$ partially surjects $Q^D_{\mathrm{RL}}(y)$ to $Q^D_{\mathrm{RL}}(zy)$, thus $|Q^D_{\mathrm{RL}}(zy)| \leq |Q^D_{\mathrm{RL}}(y)|$; in addition, $Q^D_{\mathrm{RL}}(z) \supseteq Q^D_{\mathrm{RL}}(zy)$.*

For $L \subseteq \Sigma^*$, we say y is *generic for M over L* if $y \in L$ and no right (resp., left) extension of y in L reduces the number of states produced on the right (left) boundary by the L-computations (R-computations) of any $D \in \mathcal{A}$ (any $D \in \mathcal{B}$):

$$y \in L \quad \text{and} \quad \begin{array}{l} (\forall yz \in L)(\forall D \in \mathcal{A})[\ |Q^D_{\mathrm{LR}}(yz)| = |Q^D_{\mathrm{LR}}(y)|\ ] \\ (\forall zy \in L)(\forall D \in \mathcal{B})[\ |Q^D_{\mathrm{RL}}(zy)| = |Q^D_{\mathrm{RL}}(y)|\ ]\,. \end{array}$$

If ϑ is a fixed generic string for M over L, every string of the form $\vartheta x \vartheta$ is called a *block*, with *infix* x. For $D \in \mathcal{A}$, we write $\alpha^D_{\vartheta,x\vartheta}$ simply as α^D_x, and note that it partially maps $Q^D_{\text{LR}}(\vartheta)$ to itself, since $Q^D_{\text{LR}}(\vartheta x\vartheta) \subseteq Q^D_{\text{LR}}(\vartheta)$ (by Fact 1). Similarly, for $D \in \mathcal{B}$ we write $\beta^D_{\vartheta x,\vartheta} : Q^D_{\text{RL}}(\vartheta) \rightharpoonup Q^D_{\text{RL}}(\vartheta)$ simply as β^D_x. The tuple

$$\big(\ (\alpha^D_x)_{D\in\mathcal{A}},\ (\beta^D_x)_{D\in\mathcal{B}}\ \big)$$

is the *inner behavior* of M on $\vartheta x\vartheta$, and satisfies the next lemma, by standard 'cut-and-paste' arguments (e.g., see [4, Lemma 3], [5, Fact 6], [10]), and the following fact, by the definition of generic string (e.g., see [5, Fact 5], [10]).

Lemma 1. (a) *If the inner behavior of M on $\vartheta x\vartheta$ consists of identities, then M decides identically on ϑ and $\vartheta x\vartheta$.* **(b)** *If the inner behaviors of M on $\vartheta x\vartheta$ and $\vartheta y\vartheta$ are identical, then M decides identically on $\vartheta x\vartheta$ and $\vartheta y\vartheta$.*

Fact 2. *On every $\vartheta x\vartheta \in L$, the inner behavior of M consists of permutations.*

The next fact (variant of [5, Facts 7–8]) says that the inner behavior on a block of the form $\vartheta x\vartheta y\vartheta$, where ϑ appears in the infix, composes the two inner behaviors for the overlapping blocks $\vartheta x\vartheta$ and $\vartheta y\vartheta$; this then generalizes to blocks of the form $\vartheta x^{(k)}\vartheta$, where the infix $x^{(k)} := x(\vartheta x)^{k-1}$ is k ϑ-separated copies of the same string. Finally, Fact 4 (variant of [4, Fact 7]) follows as an easy corollary.

Fact 3. *For all $D \in \mathcal{A}$, it is $\alpha^D_{x\vartheta y} = \alpha^D_x \circ \alpha^D_y$; hence $\alpha^D_{x^{(k)}} = (\alpha^D_x)^k$ for all $k \geq 1$. Similarly, for all $D \in \mathcal{B}$, it is $\beta^D_{x\vartheta y} = \beta^D_y \circ \beta^D_x$; hence $\beta^D_{x^{(k)}} = (\beta^D_x)^k$ for all $k \geq 1$.*

Fact 4. *If the inner behavior of M on $\vartheta x\vartheta$ consists of permutations, then for some $k \geq 1$ the inner behavior of M on $\vartheta x^{(k)}\vartheta$ consists of identities.*

2.3 Hardness Propagation

In the "*hardness propagation*" style of [6], all our witnesses are built by applying appropriate 'hardness increasing' operators to a single, well-understood, 'core' problem. Below, we first recall some of these operations along with some associated hardness propagation lemmata. We then also recall our one 'core' problem.

Let $\mathfrak{L} = (L, \tilde{L})$. The *reverse* and the *complement* of $\mathfrak{L}$ are the problems $\mathfrak{L}^{\mathsf{R}} := (L^{\mathsf{R}}, \tilde{L}^{\mathsf{R}})$ and $\neg\mathfrak{L} := (\tilde{L}, L)$. Easily, $\neg(\mathfrak{L}^{\mathsf{R}}) = (\neg\mathfrak{L})^{\mathsf{R}}$, and [4, Fact 12] holds:

Lemma 2. (a) *If no $\cup_{\text{L}}$1DFA with s-state components solves $\mathfrak{L}$, then no $\cup_{\text{R}}$1DFA with s-state components solves $\mathfrak{L}^{\mathsf{R}}$.* **(b)** *If no $\cap_{\text{L}}$1DFA with $(s{+}1)$-state components solves $\mathfrak{L}$, then no $\cup_{\text{L}}$1DFA with s-state components solves $\neg\mathfrak{L}$.*

The *conjunctive star* of $\mathfrak{L}$ is the problem of checking that a #-delimited list of instances of $\mathfrak{L}$ contains only positives; dually, the *disjunctive star* is the problem where at least one instance in the list must be positive [6, §3.1]:

$$\textstyle\bigwedge\mathfrak{L} := \big(\ \{\#x_1\#\cdots\#x_l\# \mid (\forall i)(x_i \in L)\},\ \{\#x_1\#\cdots\#x_l\# \mid (\exists i)(x_i \in \tilde{L})\}\ \big)$$
$$\textstyle\bigvee\mathfrak{L} := \big(\ \{\#x_1\#\cdots\#x_l\# \mid (\exists i)(x_i \in L)\},\ \{\#x_1\#\cdots\#x_l\# \mid (\forall i)(x_i \in \tilde{L})\}\ \big)\,,$$

where $\#x_1\#\cdots\#x_l\#$ means $l \geq 0$, each $x_i \in L \cup \tilde{L}$, and # is a fresh symbol. Easily,

$$\neg(\textstyle\bigwedge\mathfrak{L}) = \bigvee\neg\mathfrak{L} \qquad \neg(\bigvee\mathfrak{L}) = \bigwedge\neg\mathfrak{L} \qquad (\bigwedge\mathfrak{L})^{\mathsf{R}} = \bigwedge\mathfrak{L}^{\mathsf{R}} \qquad (\bigvee\mathfrak{L})^{\mathsf{R}} = \bigvee\mathfrak{L}^{\mathsf{R}}\,,$$

by the definitions. In addition, the following lemma holds [6, Lemma 3.3]:

Lemma 3. *If no s-state* 1DFA *solves* $\mathfrak{L}$*, then no* $\cap_{\text{L}}$1DFA *with s-state components solves* $\bigvee\mathfrak{L}$.

The *ordered star* $\mathfrak{L}_{\text{L}}{<}\mathfrak{L}_{\text{R}}$ of two problems $\mathfrak{L}_{\text{L}} = (L_{\text{L}}, \tilde{L}_{\text{L}})$ and $\mathfrak{L}_{\text{R}} = (L_{\text{R}}, \tilde{L}_{\text{R}})$ of disjoint promises is defined as follows [4, §7.2]: an instance is promised to be a list $x = \texttt{\#}x_1\texttt{\#}\cdots\texttt{\#}x_l\texttt{\#}$ of #-delimited instances of $\mathfrak{L}_{\text{L}}$ and $\mathfrak{L}_{\text{R}}$ where all positives of one of the problems appear before all positives of the other (note that this includes lists where at most one problem contributes positives); the task is to check that *either* both problems contribute positives and the one that places them first is $\mathfrak{L}_{\text{L}}$ *or* neither problem contributes any positives. So, in a positive x, there are x_i both from L_{L} and from L_{R}, and all those from L_{L} precede all those from L_{R}; or all x_i are in $\tilde{L}_{\text{L}} \cup \tilde{L}_{\text{R}}$. In a negative x, there are x_i both from L_{L} and from L_{R}, and all those from L_{R} precede all those from L_{L}; or exactly one of L_{L},L_{R} contributes some x_i. The next hardness propagation lemma is [4, Lemma 8]:

Lemma 4. *If no* $\cup_{\text{L}}$1DFA *with* $1{+}\binom{s}{2}$*-state components solves* $\mathfrak{L}_{\text{L}}$ *and no* $\cup_{\text{R}}$1DFA *with* $1{+}\binom{s}{2}$*-state components solves* $\mathfrak{L}_{\text{R}}$*, then no s-state* SDFA *solves* $\mathfrak{L}_{\text{L}}{<}\mathfrak{L}_{\text{R}}$.

The *membership problem* is defined over the alphabet $[h] \cup \mathbb{P}([h])$ as follows: "Given an $i \in [h]$ and an $\alpha \subseteq [h]$ (in this order), check that $i \in \alpha$." Formally:

$$\mathfrak{M} = \text{MEMBERSHIP}_h := \big(\, \{i\alpha \mid \alpha \subseteq [h] \;\&\; i \in \alpha\},\; \{i\alpha \mid \alpha \subseteq [h] \;\&\; i \in \overline{\alpha}\} \,\big)\,. \quad (3)$$

Easily, $\mathfrak{M}$ has an h-state 1DFA, but $\mathfrak{M}^{\text{R}}$ and $\neg\mathfrak{M}^{\text{R}}$ (where α precedes i) have no 1DFA with $<2^h{-}1$ states [6,4]. (In [4, Eq. (7)], $\mathfrak{M}^{\text{R}}$ is called SET NUM$_h$.)

3 From Few Reversals to Bounded Reversals

We now prove Theorem 1. We pick a 2DFA M with $r_M(n) \neq O(1)$, and show that $r_M(n) \neq o(n)$, too. Note that this is trivial if $r_M(n) = \infty$ for infinitely many n. So, the interesting case is when $r_M(n)$ is finite for all sufficiently large n.

Since $r_M(n) \neq O(1)$, every bound r admits infinitely many n with $r_M(n) \geq r$. Consider in particular $r := s{\cdot}(s{+}1)^{2s}$, for s the number of states in M. Then, for infinitely many n, some full computation c_n on some n-long input performs $\geq s{\cdot}(s{+}1)^{2s}$ reversals. Moreover, for all sufficiently large n, these c_n are halting, exactly because we are in the interesting case. Let c be one of these halting c_n.

Let $j_1 < \cdots < j_m$ be the indices of the cells where c performs its $\geq s{\cdot}(s{+}1)^{2s}$ reversals. Then $m \geq (s{+}1)^{2s}$, or else $m < (s{+}1)^{2s}$ cells would host $\geq s{\cdot}(s{+}1)^{2s}$ reversals, so some cell would host $>s$ reversals, so c would repeat a point on that cell and thus loop, a contradiction.

Now let $\overline{q}_0, \overline{q}_1, \ldots, \overline{q}_m$ be the crossing sequences of c on any $m{+}1$ boundaries that are separated by the m cells above. Since $m{+}1$ exceeds the number $(s{+}1)^{2s}$ of distinct crossing sequences in halting computations (cf. Sect. 2.1), two of the $\overline{q}_i$ must be identical. Let y be the infix between the corresponding two boundaries. Then the input is xyz, for some x,z.

We know y hosts ≥ 1 of the reversals of c, because it contains ≥ 1 of the cells indexed by the j_i. We also know, by a standard 'cut-and-paste' argument, that every full computation $c_t := \textsc{comp}_M(xy^tz)$ repeats on every copy of y every computation segment performed by c on y, including all reversals contained therein. So, every c_t performs ≥ 1 reversal on each copy of y, for a total of $\geq t$ reversals. Hence, for the infinitely many lengths $n_t := |xy^tz|$ some n_t-long input forces M to perform $\geq t = (n_t - |xz|)/|y|$ reversals. Hence, $r_M(n) \neq o(n)$.

So, Theorem 1 holds, making the inclusion $\mathsf{2D}[O(1)] \subseteq \mathsf{2D}[o(n)]$ an equality.

Concerning the inclusion $\mathsf{2D}[\mathsf{const}] \subseteq \mathsf{2D}[O(1)]$ one level down, it is tempting to suggest that it is also an equality, caused by the seemingly obvious reduction (analogous to that of Theorem 1) that every 2DFA with $O(1)$ reversals is a 2DFA with r reversals, for some r. But this suggestion is wrong (easily). The next tempting suggestion is that, although a 2DFA with $O(1)$ reversals is not *already* one with r reversals, it can be made into one, with some increase in size. Indeed:

Lemma 5. *Every s-state* 2DFA *with $O(1)$ reversals is equivalent to a $O(rs)$-state* 2DFA *with r reversals, for some r.*

Still, in this lemma, r may be super-polynomial in s (as in the 2DFA built in the proof of Theorem 1), resulting in a 2DFA too large to prove $\mathsf{2D}[\mathsf{const}] = \mathsf{2D}[O(1)]$.

4 Inside the Reversal Hierarchy

In this section we prove Theorem 2. We first introduce a new 'hardness increasing' operator and prove an associated 'hardness propagation' lemma.

The *r-th conjunctive power* of $\mathfrak{L} = (L, \tilde{L})$ is the problem of checking that a #-delimited list of exactly r instances of $\mathfrak{L}$ contains only positives:

$$\bigwedge\nolimits_r \mathfrak{L} := \left(\ \{\#x_1\# \cdots \#x_r\# \mid (\forall i)(x_i \in L)\}, \ \ \{\#x_1\# \cdots \#x_r\# \mid (\exists i)(x_i \in \tilde{L})\} \ \right),$$

where $\#x_1\# \cdots \#x_r\#$ means that every $x_i \in L \cup \tilde{L}$ and # is a fresh symbol.

Lemma 6. *If no $4rs^{2r+1}$-state* SDFA *solves $\mathfrak{L}$, then no s-state* 2DFA *with $<r$ reversals solves $\bigwedge_r \mathfrak{L}$.*

Proof. Let $\mathfrak{L} = (L, \tilde{L})$. Let M be an s-state 2DFA with $<r$ reversals for $\bigwedge_r \mathfrak{L}$. We build a SDFA M' for $\mathfrak{L}$ with $4rs^{2r+1}$ states. We first introduce *certificates*, then show how they characterize the positives of $\mathfrak{L}$, then use them to design M'.

Pick any positive x of $\mathfrak{L}$. Then $w := \#(x\#)^r$ is a positive of $\bigwedge_r \mathfrak{L}$. Therefore, $c := \textsc{comp}_M(w)$ is accepting. Moreover, the reversals in c are fewer than the copies of x in w. So, on one or more of these copies, c performs 0 reversals. Fix any such copy (e.g., the leftmost one). On it, c consists of $t \leq r$ one-way traversals (one-way, since there are 0 reversals; and $\leq r$, because with $<r$ reversals in total c can traverse each infix $\leq r$ times). Let $\overline{p}_x := (p_1, \ldots, p_t)$ and $\overline{q}_x := (q_1, \ldots, q_t)$ be the crossing sequences of c on the outer boundaries of that copy of x (Fig. 1b). Finally, consider the set of all pairs of crossing sequences created in this way,

$$\mathcal{C} := \{(\overline{p}_x, \overline{q}_x) \mid x \in L\},$$

as we iterate over all positives of $\mathfrak{L}$. We use this set in the next definition.

Definition. A pair $(\overline{p},\overline{q})$ of t-long sequences of states of M is a *certificate* for an instance x of $\mathfrak{L}$ if it satisfies the following three clauses:

1. $(\overline{p},\overline{q}) \in \mathcal{C}$.
2. For every odd $i = 1,\dots,t$: $\textsc{lcomp}_{M,p_i}(x)$ is one-way and hits right into q_i.
3. For every even $i = 1,\dots,t$: $\textsc{rcomp}_{M,q_i}(x)$ is one-way and hits left into p_i.

Claim. An instance of $\mathfrak{L}$ is positive iff it has a certificate.

Proof. [⇒] Let $x \in L$. Then clearly $(\overline{p}_x,\overline{q}_x)$ is a certificate for x. [⇐] Let $\tilde{x} \in \tilde{L}$. Suppose $\tilde{x}$ has a certificate $(\overline{p},\overline{q})$. By Clause 1, there is $x \in L$ such that the accepting computation $c := \textsc{comp}_M(\#(x\#)^r)$ exhibits $\overline{p}$ and $\overline{q}$ on the outer boundaries of a copy of x on which it contains 0 reversals. By Clauses 2 and 3, M notices no difference if we replace that copy with a copy of $\tilde{x}$. So, the computation of M on the modified string is also accepting. But this modified string is a negative of $\bigwedge_r\mathfrak{L}$. Therefore, M does not solve $\bigwedge_r\mathfrak{L}$ —a contradiction. ⊡

By the Claim, one way to check an instance x of $\mathfrak{L}$ is to check whether any pair in $\mathcal{C}$ is a certificate for it; because $\mathcal{C}$ is 'small' and each pair is checkable by 'few' sweeps, this strategy can be implemented by a 'small' SDFA. Specifically, M' iterates over all $((p_1,\dots,p_t),(q_1,\dots,q_t)) \in \mathcal{C}$. For each of them and each odd (resp., even) $i = 1,\dots,t$, it simulates M on x from p_i (from q_i) on the leftmost (rightmost) symbol, to see whether it hits right (left) into q_i (into p_i) without ever reversing; on any attempt to reverse, M' stops simulating and just completes the sweep. If these checks succeed for all i, then M' accepts; otherwise, it continues to the next pair. If all pairs have been tried, then M' rejects.

If Q are the states of M, then M' uses states $Q' := \mathcal{C}\times\{1,\dots,r\}\times Q_\perp$. State $(\overline{p},\overline{q},i,p)$ means we are at state p in simulating M in the i-th check for the candidate certificate $(\overline{p},\overline{q})$; if $p = \perp$, then the i-th check has already failed due to an attempt to reverse, and we are just completing the sweep. Easily, $|\mathcal{C}| \leq \sum_{t=0}^{r}(s^t\cdot s^t) \leq 2s^{2r}$, therefore $|Q'| = |\mathcal{C}|\cdot r\cdot(|Q|+1) \leq 2s^{2r}\cdot r\cdot(s+1) \leq 4rs^{2r+1}$. □

We are now ready to introduce our witness. For $r \geq 1$ and $\mathfrak{M}$ as in (3), it is

$$\mathfrak{R}_r := \textstyle\bigwedge_r \big[\, (\bigwedge\mathfrak{M}^{\mathrm{R}}) < (\bigwedge\mathfrak{M}) \,\big]\,. \tag{4}$$

So, an instance of $\mathfrak{R}_r$ is a list of the form $\$y_1\$\cdots\$y_r\$$; each y_j is a list of the form $*x_1*\cdots*x_l*$, for arbitrary l; and each x_j is a list of the form $\#\alpha_1 i_1\#\cdots\#\alpha_l i_l\#$ or $\#i_1\alpha_1\#\cdots\#i_l\alpha_l\#$, again for arbitrary l. The task is to check that, in every y_j: *either* every x_j has some i_j not in the adjacent α_j (i.e., all x_j are negatives of $\bigwedge\mathfrak{M}^{\mathrm{R}}$ and $\bigwedge\mathfrak{M}$); *or* x_j of both forms exist with all their i_j in the adjacent α_j, and those of the set-number form precede those of the number-set form (i.e., both $\bigwedge\mathfrak{M}^{\mathrm{R}}$ and $\bigwedge\mathfrak{M}$ contribute positives, and those of $\bigwedge\mathfrak{M}^{\mathrm{R}}$ precede those of $\bigwedge\mathfrak{M}$).

For the lower bound, we know that every SDFA for $\bigwedge\mathfrak{M}^{\mathrm{R}}<\bigwedge\mathfrak{M}$ has $2^{\Omega(h)}$ states (by the lower-bound argument of [4, §7.3], which uses Lemma 4). Therefore, by Lemma 6, every 2DFA with $< r$ reversals for $\mathfrak{R}_r$ has $2^{\Omega(h/r)}$ states.

For the upper bound, we start as in [4, §7.3]. We let M_0 be the h-state 1DFA for $\mathfrak{M}$. We then build a $O(h)$-state 1DFA M_1 for $\bigwedge\mathfrak{M}$, which just repeatedly simulates M_0 on the successive instances of $\mathfrak{M}$ and accepts iff all are accepted.

Next, we build a $O(h)$-state 2DFA M_2 with 1 reversal for $\bigwedge\mathfrak{M}^{\mathrm{R}}<\bigwedge\mathfrak{M}$. On input $*x_1*\cdots*x_l*$, M_2 scans forward simulating M_1 on every instance of $\bigwedge\mathfrak{M}$ until it detects a positive or reaches $\dashv$. In either case, it reverses and scans backwards simulating M_1 on (the reverse of) every instance of $\bigwedge\mathfrak{M}^{\mathrm{R}}$ until it detects a positive or reaches $\vdash$. Then M_2 knows what to do: (1) if neither scan detected a positive, then all x_j are negative, so M_2 must accept; (2) if the forward scan detected no positive but the backward scan did, then only $\bigwedge\mathfrak{M}^{\mathrm{R}}$ contributes positives, so M_2 must reject; (3) if the forward scan detected a positive but the backward scan did not, then M_2 must reject either because only $\bigwedge\mathfrak{M}$ contributes positives or because both problems do but the order is wrong; (4) if both scans detected a positive, then both problems contribute and the order is correct, so M_2 must accept. So, M_2 finishes the backward scan (if needed) and decides on $\vdash$.

Finally, we build a 2DFA R_r with r reversals for $\mathfrak{R}_r$. On input $\$y_1\$\cdots\$y_r\$$, a successive pair $\$y_j\$y_{j+1}\$$ is checked by a 2-reversal LR-traversal, as follows: scan forward past y_j; simulate M_2 on y_{j+1} by a 1-reversal L-turn which ends on the middle $\$$; from there, simulate M_2 on (the reverse of) y_j by a 1-reversal R-turn which also ends on the middle $\$$; from there, scan forward past y_{j+1}. Easily, this check needs $O(h)$ states. Now, if r is even, then R_r simply repeats this check on every pair of successive y_j until it reaches $\dashv$. If r is odd, then R_r first scans forward past $y_1,\ldots,y_{r-1}$, to simulate M_2 on y_r by a 1-reversal L-turn that ends on the penultimate $\$$; from there, it starts checking pairs of successive y_j by repeating the above check (backwards and in reverse) until $\vdash$. Easily, the number of states in R_r is $O(r+h)$ —for even r, it is only $O(h)$.

5 Inside the Inner-Reversal Hierarchy

We now prove Theorem 3. Most crucially, we improve the lower bound of Sect. 4 to make it (i) independent of r, and (ii) valid even when only *inner* reversals are restricted. For this, we enhance our chain of hardness propagation, by proving variants of Lemmata 4 and 6 where SDFAs are replaced by $\mathrm{P}_2$1DFAs.

Lemma 4*. *If no* $\cup_{\mathrm{L}}$1DFA *with* $1+\binom{s}{2}$*-state components solves* $\mathfrak{L}_{\mathrm{L}}$ *and no* $\cup_{\mathrm{R}}$1DFA *with* $1+\binom{s}{2}$*-state components solves* $\mathfrak{L}_{\mathrm{R}}$*, then no* $\mathrm{P}_2$1DFA *with* s*-state components solves* $\mathfrak{L}_{\mathrm{L}}<\mathfrak{L}_{\mathrm{R}}$.

Proof. The structure of the argument is exactly as in the proof of [4, Lemma 8]; we just adapt some of its steps for $\mathrm{P}_2$1DFAs. So, let $\mathfrak{L}_{\mathrm{L}}=(L_{\mathrm{L}},\tilde{L}_{\mathrm{L}})$, $\mathfrak{L}_{\mathrm{R}}=(L_{\mathrm{R}},\tilde{L}_{\mathrm{R}})$. Suppose some $\mathrm{P}_2$1DFA $M=(\mathcal{A},\mathcal{B},F)$ solves $\mathfrak{L}_{\mathrm{L}}<\mathfrak{L}_{\mathrm{R}}$ with s-state components.

We first consider the strings of #-delimited instances of $\mathfrak{L}_{\mathrm{L}}$ and $\mathfrak{L}_{\mathrm{R}}$ where neither problem contributes positives, and those where exactly one does:

$$L := \{\#x_1\#\cdots\#x_l\# \mid (\forall i)(x_i\in\tilde{L}_{\mathrm{L}}\cup\tilde{L}_{\mathrm{R}})\}\text{, and}$$

$$\tilde{L} := \{\#x_1\#\cdots\#x_l\# \mid (\exists i)(x_i\in L_{\mathrm{L}}\cup L_{\mathrm{R}})\ \ \&\ \ \neg(\exists i)(\exists j)(x_i\in L_{\mathrm{L}}\ \&\ x_j\in L_{\mathrm{R}})\}\,,$$

where $\#x_1\#\cdots\#x_l\#$ means $l\geq 0$ and every $x_i\in L_{\mathrm{L}}\cup\tilde{L}_{\mathrm{L}}\cup L_{\mathrm{R}}\cup\tilde{L}_{\mathrm{R}}$. Note that all strings in $L\cup\tilde{L}$ are instances of $\mathfrak{L}_{\mathrm{L}}<\mathfrak{L}_{\mathrm{R}}$: positive if in L, negative if in $\tilde{L}$. So,

M solves $(L, \tilde{L})$. From now on, fix ϑ to be a generic string for M over L. (The existence of such a string follows from standard observations [6, §3.3.2].)

Definition. A pair $\{p, q\}$ of distinct states in M is a *forward certificate* for an instance x of $\mathfrak{L}_{\mathrm{L}}$ or $\mathfrak{L}_{\mathrm{R}}$ if there exists $D \in \mathcal{A}$ such that

$$p, q \in Q^{D}_{\mathrm{LR}}(\vartheta) \quad \text{and} \quad \begin{array}{l}\text{if both } \textsc{lcomp}_{D,p}(x\vartheta) \text{ and } \textsc{lcomp}_{D,q}(x\vartheta) \text{ hit right,}\\ \text{then they do so into the same state.}\end{array} \tag{5}$$

A *backward certificate* is defined symmetrically, with $\mathcal{A}$, Q^{D}_{LR}, $\textsc{lcomp}_{D,\,\cdot}\,(x\vartheta)$, and "hit right" replaced respectively by $\mathcal{B}$, Q^{D}_{RL}, $\textsc{rcomp}_{D,\,\cdot}\,(\vartheta x)$, and "hit left".

Claim 1. An instance of $\mathfrak{L}_{\mathrm{L}}$ or $\mathfrak{L}_{\mathrm{R}}$ is positive iff it has a certificate.

Proof. As in [4, Lemma 8]. [$\Rightarrow$] By Fact 4 and Lemma 1a. [$\Leftarrow$] By Fact 2. ⊡

Note that, for positive instances, Claim 1 does not specify whether the existing certificates are of the forward or of the backward kind. It turns out that a stronger criterion is possible for at least one of $\mathfrak{L}_{\mathrm{L}}$ or $\mathfrak{L}_{\mathrm{R}}$.

Claim 2. At least one is true: (i) every positive instance of $\mathfrak{L}_{\mathrm{L}}$ has a forward certificate, or (ii) every positive instance of $\mathfrak{L}_{\mathrm{R}}$ has a backward certificate.

Proof. Suppose not. Then there is $x \in L_{\mathrm{L}}$ with no forward certificate and $y \in L_{\mathrm{R}}$ with no backward certificate. As in the proof of Claim 1, this means that every α^{D}_{x} for $D \in \mathcal{A}$ permutes $Q^{D}_{\mathrm{LR}}(\vartheta)$ and every β^{D}_{y} for $D \in \mathcal{B}$ permutes $Q^{D}_{\mathrm{RL}}(\vartheta)$. Pick $k \geq 1$ so that each of these $|\mathcal{A}|+|\mathcal{B}|$ permutations becomes an identity after k iterations:

$$(\forall D \in \mathcal{A})[\ (\alpha^{D}_{x})^{k} = \mathrm{id}\ ] \qquad \text{and} \qquad (\forall D \in \mathcal{B})[\ (\beta^{D}_{y})^{k} = \mathrm{id}\ ]\,,$$

where 'id' is the identity function on the appropriate domain. Then, by Fact 3,

$$(\forall D \in \mathcal{A})[\ \alpha^{D}_{x^{(k)}} = \mathrm{id}\ ] \qquad \text{and} \qquad (\forall D \in \mathcal{B})[\ \beta^{D}_{y^{(k)}} = \mathrm{id}\ ]\,. \tag{6}$$

Intuitively, this means that no $D \in \mathcal{A}$ can distinguish $\vartheta x^{(k)} \vartheta$ from ϑ, and no $D \in \mathcal{B}$ can distinguish $\vartheta y^{(k)} \vartheta$ from ϑ. Hence, M cannot distinguish between

$$\vartheta x^{(k)} \vartheta y^{(k)} \vartheta \qquad \text{and} \qquad \vartheta y^{(k)} \vartheta x^{(k)} \vartheta\,, \tag{7}$$

because they both 'look' like $\vartheta y^{(k)} \vartheta$ to every $D \in \mathcal{A}$, and like $\vartheta x^{(k)} \vartheta$ to every $D \in \mathcal{B}$. If this intuition is correct, then M treats identically a positive (on the left) and a negative (on the right) instance of $\mathfrak{L}_{\mathrm{L}} {<} \mathfrak{L}_{\mathrm{R}}$—a contradiction.

Indeed, the inner behavior of every $D \in \mathcal{A}$ on the two instances of (7) is:

$$\begin{aligned} \alpha^{D}_{x^{(k)}\vartheta y^{(k)}} &= \alpha^{D}_{x^{(k)}} \circ \alpha^{D}_{y^{(k)}} = \mathrm{id} \circ \alpha^{D}_{y^{(k)}} = \alpha^{D}_{y^{(k)}} \\ \alpha^{D}_{y^{(k)}\vartheta x^{(k)}} &= \alpha^{D}_{y^{(k)}} \circ \alpha^{D}_{x^{(k)}} = \alpha^{D}_{y^{(k)}} \circ \mathrm{id} = \alpha^{D}_{y^{(k)}}\,, \end{aligned}$$

where in each line all functions are partial from $Q^{D}_{\mathrm{LR}}(\vartheta)$ to itself, the first step uses Fact 3, and the second step uses (6). Hence, $\alpha^{D}_{x^{(k)}\vartheta y^{(k)}} = \alpha^{D}_{y^{(k)}} = \alpha^{D}_{y^{(k)}\vartheta x^{(k)}}$. By this and a symmetric argument for every $D \in \mathcal{B}$, we eventually conclude that

$$\left.\begin{array}{l} \big(\ (\alpha^{D}_{x^{(k)}\vartheta y^{(k)}})_{D\in\mathcal{A}},\ (\beta^{D}_{x^{(k)}\vartheta y^{(k)}})_{D\in\mathcal{B}}\ \big) \\ \big(\ (\alpha^{D}_{y^{(k)}\vartheta x^{(k)}})_{D\in\mathcal{A}},\ (\beta^{D}_{y^{(k)}\vartheta x^{(k)}})_{D\in\mathcal{B}}\ \big) \end{array}\right\} = \big(\ (\alpha^{D}_{y^{(k)}})_{D\in\mathcal{A}},\ (\beta^{D}_{x^{(k)}})_{D\in\mathcal{B}}\ \big)\,.$$

Hence, M treats the blocks of (7) the same (Lemma 1b), as expected. ⊡

Now, if Claim 2i is true, then along with Claim 1 it implies a criterion for $\mathfrak{L}_{\mathrm{L}}$: *an instance of* $\mathfrak{L}_{\mathrm{L}}$ *is positive iff it has a forward certificate.* We thus get:

Claim 3. Some $\cup_{\mathrm{L}}$1DFA with $1+\binom{s}{2}$-state components solves $\mathfrak{L}_{\mathrm{L}}$.

Proof. By the criterion, an instance x of $\mathfrak{L}_{\mathrm{L}}$ is positive iff there is $D \in \mathcal{A}$ and distinct $p, q \in Q^{D}_{\mathrm{LR}}(\vartheta)$ such that *either* one of $\textsc{lcomp}_{D,p}(x\vartheta)$ or $\textsc{lcomp}_{D,q}(x\vartheta)$ hangs *or* both hit right into the same state. A $\cup_{\mathrm{L}}$1DFA can check this using a $1+\binom{s}{2}$-state component $D_{p,q}$ for every such combination of D and p, q. ⊡

If Claim 2ii holds, we work similarly with $\mathfrak{L}_{\mathrm{R}}$ and backward certificates. □

Lemma 6*. *If no* P$_2$1DFA *with s-state components solves $\mathfrak{L}$, then no s-state* 2DFA *with $<r$ inner reversals solves $\bigwedge_r \mathfrak{L}$.*

Proof. Let $\mathfrak{L} = (L, \tilde{L})$. Let M be a 2DFA with $<r$ inner reversals for $\bigwedge_r \mathfrak{L}$, with set of states $Q = [s]$. We build a P$_2$1DFA M' with s-state components for $\mathfrak{L}$.

We use *certificates* as in Lemma 6. For each $x \in L$, $c := \textsc{comp}_M(\#(x\#)^r)$ is accepting and avoids reversals on one or more copies of x (since every reversal on a copy of x is inner). So, the crossing sequences $\overline{p}_x$,$\overline{q}_x$ on the outer boundaries of the leftmost such copy are again 'linked' by t one-way traversals (Fig. 1b). This time, however, it is not guaranteed that $t \le r$, as some pairs of successive traversals may be separated by *outer* reversals, whose number is not bounded by r. We just know that $t \le 2s$ (or else c would repeat a state on an outer cell of x, and loop), so the set $\mathcal{C} := \{(\overline{p}_x, \overline{q}_x) \mid x \in L\}$ of candidate certificates may be exponentially large, forbidding an exhaustive search by a small SDFA.

However, a small-component P$_2$1DFA can delegate this exhaustive search to its set of accepting tuples. So, we let $M' := (\{A_p \mid p \in Q\}, \{B_p \mid p \in Q\}, F)$. Each 1DFA A_p simulates M from p for as long as it moves right; if M ever attempts to reverse, A_p hangs. Similarly, each B_p simulates M from p for as long as it moves left, and hangs at any attempt to reverse. Hence, on input x, M' simulates M from every state and in every fixed direction, covering every possible one-way traversal of x by M. In the end, it checks whether x has a certificate by comparing the results of these $2s$ computations against each $(\overline{p}, \overline{q}) \in \mathcal{C}$. Formally, for each $\overline{p} = (p_1, \dots, p_t)$ and $\overline{q} = (q_1, \dots, q_t)$ we let $F_{(\overline{p},\overline{q})}$ be the set of all $2s$-tuples that we can build from two copies of all states in $Q = \{0, 1, \dots, s{-}1\}$

$$(\; 0, 1, \dots, s{-}1, \;\; 0, 1, \dots, s{-}1 \;),$$

by replacing (i) every odd-indexed p_i in the left copy with the respective q_i (to ask A_{p_i} to hit right into q_i); (ii) every even-indexed q_i in the right copy with the respective p_i (to ask B_{q_i} to hit left into p_i) and (iii) all other states in either copy with any result in $Q_\perp$ (to let all other 1DFAs free). This way, $F_{(\overline{p},\overline{q})}$ is all tuples which prove that $(\overline{p}, \overline{q})$ is a certificate. So, letting $F := \bigcup_{(\overline{p},\overline{q}) \in \mathcal{C}} F_{(\overline{p},\overline{q})}$, we ensure that M' accepts x iff x has a certificate, namely iff $x \in L$. □

We are now ready to proceed to the main argument that proves Theorem 3.

For the lower bound, we start as in [4, §7.3]. We know no (2^h-2)-state 1DFA solves $\neg\mathfrak{M}^{\mathrm{R}}$. So, Lemma 3 implies no $\cap_{\mathrm{L}}$1DFA with (2^h-2)-state components solves $\bigvee\neg\mathfrak{M}^{\mathrm{R}}$. Hence, Lemma 2b for $\bigvee\neg\mathfrak{M}^{\mathrm{R}} = \neg\bigwedge\mathfrak{M}^{\mathrm{R}}$ implies that

no $\cup_{\mathrm{L}}$1DFA with (2^h-3)-state components solves $\bigwedge\mathfrak{M}^{\mathrm{R}}$.

This, together with Lemma 2a for $\bigwedge\mathfrak{M}^{\mathrm{R}} = (\bigwedge\mathfrak{M})^{\mathrm{R}}$, implies that

no $\cup_{\mathrm{R}}$1DFA with (2^h-3)-state components solves $\bigwedge\mathfrak{M}$.

So, by Lemma 4*, in every $\mathrm{P}_2$1DFA for $\bigwedge\mathfrak{M}^{\mathrm{R}}{<}\bigwedge\mathfrak{M}$ some component has $\Omega(2^{h/2})$ states. By Lemma 6*, the same holds for all 2DFAs with $<r$ inner reversals for $\mathfrak{R}_r$.

For the upper bound, we note that our 2DFA R_r from Sect. 4 performs only inner reversals. Moreover, its size can stay independent of r, if we allow ≤ 1 outer reversal: for odd r, we modify R_r to work as if r were even; this causes 1 outer reversal during the check of y_r. So, the modified R_r solves $\mathfrak{R}_r$ with $O(h)$ states, r inner reversals, and 0 or 1 outer reversals (depending on the parity of r).

6 Conclusion

We studied 2DFAs with *few*, *bounded*, and *fixed* reversals ($o(n)$, $O(1)$, r, respectively). We showed that the first two are actually the same, whereas small 2DFAs of the last kind strictly increase their power with every additional reversal, even if we focus only on those performed strictly between the end-markers.

It would have been nice if we had also resolved $\mathsf{2D}[\mathsf{const}] \subseteq \mathsf{2D}[O(1)]$. It would also be interesting to repeat this analysis for 2NFAs [3, Research Problem 4].

References

1. Balcerzak, M., Niwiński, D.: Two-way deterministic automata with two reversals are exponentially more succinct than with one reversal. Information Processing Letters 110, 396–398 (2010)
2. Geffert, V., Mereghetti, C., Pighizzini, G.: Complementing two-way finite automata. Information and Computation 205(8), 1173–1187 (2007)
3. Hromkovič, J.: Descriptional complexity of finite automata: concepts and open problems. Journal of Automata, Languages and Combinatorics 7(4), 519–531 (2002)
4. Kapoutsis, C.: Nondeterminism is essential in small two-way finite automata with few reversals. Information and Computation (to appear)
5. Kapoutsis, C.: Deterministic moles cannot solve liveness. Journal of Automata, Languages and Combinatorics 12(1-2), 215–235 (2007)
6. Kapoutsis, C., Královič, R., Mömke, T.: Size complexity of rotating and sweeping automata. Journal of Computer and System Sciences 78(2), 537–558 (2012)
7. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Proceedings of FOCS, pp. 188–191 (1971)
8. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Proceedings of STOC, pp. 275–286 (1978)
9. Sipser, M.: Halting space-bounded computations. Theoretical Computer Science 10, 335–338 (1980)
10. Sipser, M.: Lower bounds on the size of sweeping automata. Journal of Computer and System Sciences 21(2), 195–202 (1980)

Strictness of the Collapsible Pushdown Hierarchy*

Alexander Kartzow[1] and Paweł Parys[2]

[1] Universität Leipzig,
Johannisgasse 26, 04103 Leipzig, Germany
kartzow@informatik.uni-leipzig.de
[2] University of Warsaw,
ul. Banacha 2, 02-097 Warszawa, Poland
parys@mimuw.edu.pl

Abstract. We present a pumping lemma for each level of the collapsible pushdown graph hierarchy in analogy to the second author's pumping lemma for higher-order pushdown graphs (without collapse). Using this lemma, we give the first known examples that separate the levels of the collapsible pushdown graph hierarchy and of the collapsible pushdown tree hierarchy, i.e., the hierarchy of trees generated by higher-order recursion schemes. This confirms the open conjecture that higher orders allow one to generate more graphs and more trees.

Full proofs can be found in the arXiv version[10] of this paper.

1 Introduction

Already in the 70's, Maslov ([12,13]) generalised the concept of pushdown systems to higher-order pushdown systems and studied such devices as acceptors of string languages. In the last decade, renewed interest in these systems has arisen. They are now studied as generators of graphs and trees. Knapik et al. [11] showed that the class of trees generated by deterministic level n pushdown systems coincides with the class of trees generated by *safe* level n recursion schemes,[1] and Caucal [5] gave another characterisation: trees on level $n+1$ are obtained from trees on level n by an MSO-interpretation followed by an unfolding. Carayol and Wöhrle [4] studied the ε-contractions of configuration graphs of level n pushdown systems and proved that these are exactly the graphs in the n-th level of the Caucal hierarchy.

Driven by the question whether safety implies a semantical restriction to recursion schemes, Hague et al. [7] extended the model of higher-order pushdown systems by introducing a new stack operation called collapse. They showed that the trees generated by the resulting collapsible pushdown systems coincide exactly with the class of trees generated by all higher-order recursion schemes and

* The first author is supported by the DFG project "GELO". The second author is partially supported by the Polish Ministry of Science grant nr N N206 567840. The collaboration of the authors is supported by the ESF project "GAMES".

[1] Safety is a syntactic restriction on the recursion scheme.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 566–577, 2012.

this correspondence is level-by-level. Recently, Parys ([14,15]) proved the safety conjecture, i.e., he showed that higher-order recursion schemes generate more trees than safe higher-order recursion schemes, which implies that the class of collapsible pushdown trees is a proper extension of the class of higher-order pushdown trees. Similarly, due to their different behaviour with respect to MSO model checking, we know that the class of collapsible pushdown graphs forms a proper extension of the class of higher-order pushdown graphs.

Several questions concerning the relationship of these classes have been left open so far. Up to now it was not known whether collapsible pushdown graphs form a strict hierarchy in the sense that for each $n \in \mathbb{N}$ the class of level n collapsible pushdown graphs is strictly contained in the class of level $(n+1)$ collapsible pushdown graphs. The same question was open for the hierarchy of trees generated by collapsible pushdown systems (i.e. by recursion schemes). Extending the pumping arguments of Parys for higher-order pushdown systems [16] to the collapsible pushdown setting, we answer both questions in the affirmative.

Our main technical contribution is the following *pumping lemma* which subsumes the pumping lemmas for level 2 collapsible pushdown systems [9] and for higher-order pushdown systems [16]. Set $\exp_0(i) = i$ and $\exp_{k+1}(i) = 2^{\exp_k(i)}$.

Theorem 1.1. *Let $\mathcal{S}$ be a collapsible pushdown system of level n. Let $\mathcal{G}$ be the ε-contraction of the configuration graph of $\mathcal{S}$. Assume that it is finitely branching and that there is a path in $\mathcal{G}$ of length m from the initial configuration to some configuration c. For $C_{\mathcal{S}}$ a constant only depending on $\mathcal{S}$, if there is a path p in $\mathcal{G}$ of length at least $\exp_{n-1}((m+1)\cdot C_{\mathcal{S}})$ which starts in c, then there are infinitely many paths in $\mathcal{G}$ which start in c and end in configurations having the same control state as the last configuration of p.*

Corollary 1.2. *Let $\mathcal{G}$ be the successor tree induced by $\{1^i 0^{\exp_n(i)} \mid i \in \mathbb{N}\}$.*

$\mathcal{G}$ is the ε-contraction of the configuration graph of a pushdown system of level $n+1$ but not the ε-contraction of the configuration graph of any collapsible pushdown system of level n. Moreover, $\mathcal{G}$ is generated by a safe level $(n+1)$ recursion scheme but not by any level n recursion scheme.

$\mathcal{G}$ is not in the n-th level because application of the pumping lemma to the node $1^{2\cdot C_{\mathcal{S}}}0$ yields a contradiction. The proof that $\mathcal{G}$ is in level $n+1$ follows from [2]. Moreover, our techniques allow us to decide the following problems.[2]

Lemma 1.3. *Given a collapsible pushdown system, it is decidable*

1. *whether the ε-contraction of its configuration graph is finitely branching,*
2. *whether the ε-contraction of its configuration graph is finite, and*
3. *whether the unfolding of the ε-contraction of its configuration graph is finite.*

[2] We thank several anonymous referees of our LICS submissions for pointing our interest towards these problems.

1.1 Related Work

Hayashi [8] and Gilman [6] proved a pumping and a shrinking lemma for indexed languages. It is shown in [1] that indexed languages are exactly the string languages accepted by level 2 collapsible pushdown systems. For higher levels, no shrinking techniques are known so far. Since our pumping lemma can be used only for finitely branching systems, it cannot be used to show that certain string languages do not occur on certain levels of the (collapsible) higher-order pushdown hierarchy. Note that we do not know whether the string languages accepted by nondeterministic level n pushdown systems and by nondeterministic level n collapsible pushdown systems coincide for $n > 2$. Thus, it is an interesting open question whether there is a stronger pumping lemma for runs of higher-order systems that could be used to separate these classes of string languages.

2 Collapsible Pushdown Graphs

Collapsible pushdown systems of level n (from now on $n \in \mathbb{N}$ is fixed) are an extension of pushdown systems where we replace the stack by an n-fold nested stack structure. This higher-order stack is manipulated using a a push, a pop and a collapse operation for each stack level $1 \leq i \leq n$. When a new symbol is pushed onto the stack, we attach a copy of a certain level k substack of the current stack to this symbol (for some $1 \leq k \leq n$) and at some later point the collapse operation may replace the topmost level k stack with the level k stack stored in the topmost symbol of the stack (we also talk about the linked k-stack of the topmost symbol). In some weak sense the collapse operation allows one to jump back to the (level k) stack where the current topmost symbol was created for the first time.

Definition 2.1. *Given a number n (the level of the system) and stack alphabet Γ, we define the set of stacks as the smallest set satisfying the following.*

- *If $s_1, s_2, \ldots, s_m$ are $(k-1)$-stacks, where $1 \leq k \leq n$, then the sequence $[s_1, s_2, \ldots, s_m]$ is a k-stack (with s_m its topmost $k-1$-stack). This includes the empty sequence ($m = 0$).*
- *If s^k is a k-stack, where $1 \leq k \leq n$, and $\gamma \in \Gamma$, then (γ, k, s^k) is a 0-stack.*

For a 0-stack $s^0 = (\gamma, k, t^k)$ we call γ the symbol *of s^0 and for some k-stack t^k the* topmost symbol *is the symbol of its topmost 0-stack.*

For a k-stack s^k and a $(k-1)$-stack s^{k-1} we write $s^k : s^{k-1}$ to denote the k-stack obtained by appending s^{k-1} on top of s^k. We write $s^2 : s^1 : s^0$ for $s^2 : (s^1 : s^0)$.

Let us remark that in the original definition stacks are defined differently: they are not nested, a 0-stack does not store the linked k-stack but the number of pop-operations a collapse is equivalent to. With respect to constructible stacks this is only a syntactical difference. Independently, Broadbent et al. recently also introduced this representation of stacks under the name *annotated stacks* in [3].

Definition 2.2. *We define the set of stack operations OP as follows. We decompose a stack s of level n into its topmost stacks $s^n : s^{n-1} : \cdots : s^0$. For $s^i \neq \emptyset$ set $\mathsf{pop}^i(s) := s^n : \cdots : s^{i+1} : s^i$. For $s^i = \emptyset$, $\mathsf{pop}^i(s)$ is undefined. For $2 \leq i \leq n$ we have $\mathsf{push}^i(s) := s^n : \cdots : s^{i+1} : (s^i : \cdots : s^0) : s^{i-1} : \cdots : s^0$. The level 1 push is $\mathsf{push}^1_{\gamma,k}$ for $\gamma \in \Gamma$, $1 \leq k \leq n$ which is defined by $\mathsf{push}^1_{\gamma,k}(s) := s^n : \cdots : s^2 : (s^1 : s^0) : (\gamma, k, s^k)$.*[3] *The collapse operation col^i (where $1 \leq i \leq n$) is defined if the topmost 0-stack is (γ, i, t^i), and t^i is not empty. Then it is $\mathsf{col}^i(s) := s^n : \cdots : s^{i+1} : t^i$. Otherwise the collapse operation is undefined.*

Definition 2.3. *The* initial 0-stack $\bot_0$ *is $(\bot, n, [])$ for a special symbol $\bot \in \Gamma$, i.e., a 0-stack only containing the symbol $\bot$ with link to the empty stack. The* initial $(k+1)$-stack *is $[\bot_k]$. Some n-stack s is a* pushdown store *(or* pds*), if there is a finite sequence of stack operations that create s from $\bot_n$.*

Remark 2.4. If s is a pds and if $\mathsf{col}^j(s)$ is defined, then there is a $k \geq 1$ such that $\mathsf{col}^j(s)$ is the stack obtained from s by applying pop^j k times.

Definition 2.5. *A* collapsible pushdown system of level n *(an n-CPS) is a tuple $\mathcal{S} = (\Gamma, A, Q, q_I, \bot, \Delta)$ where Γ is a finite stack alphabet with $\bot \in \Gamma$, A is a finite input alphabet, Q is a finite set of states, $q_I \in Q$ is an initial state, and $\Delta \subseteq Q \times \Gamma \times (A \cup \{\varepsilon\}) \times Q \times OP$ is a transition relation. A* configuration *is a pair (q, s) with $q \in Q$ and s a pds. The* initial configuration of $\mathcal{S}$ *is $(q_I, \bot_n)$.*

Definition 2.6. *We define a* run *of a CPS $\mathcal{S}$. For $0 \leq i \leq m$, let $c_i = (q_i, s_i)$ be a configuration of $\mathcal{S}$ and let γ_i denote the topmost stack symbol of s_i. A run R of length m from c_0 to c_m is a sequence $c_0 \vdash^{a_1} c_1 \vdash^{a_2} \cdots \vdash^{a_m} c_m$ such that, for $1 \leq i \leq m$, there is a transition $(q_{i-1}, \gamma_{i-1}, a_i, q_i, op)$ where $s_i = op(s_{i-1})$. We set $R(i) := c_i$ and call $|R| := m$ the* length *of R. The* subrun *$R{\restriction}_{i,j}$ is $c_i \vdash^{a_{i+1}} c_{i+1} \vdash^{a_{i+2}} \cdots \vdash^{a_j} c_j$. For runs R, S with $R(|R|) = S(0)$, we write $R \circ S$ for the* composition *of R and S which is defined as expected.*

Definition 2.7. *Let $\mathcal{S}$ be a collapsible pushdown system. The* (collapsible pushdown) graph[4] *of $\mathcal{S} = (\Gamma, A, Q, q_I, \bot, \Delta)$ is $\mathcal{G} := (G, (E_a)_{a \in A \cup \{\varepsilon\}})$ where G consists of all configurations reachable from $(q_0, \bot_n)$ and there is an a-labelled edge from a configuration c to a configuration d if there is a run $c \vdash^a d$. The* ε-contraction *of $\mathcal{G}$ is the graph $(G', (E'_a)_{a \in A})$ where $G' := \{c \in G : \exists d \in G\ d \vdash^a c$ for some $a \in A\}$ and two configurations c, d are connected by E'_a if there is a run $c \vdash^\varepsilon c_1 \vdash^\varepsilon \cdots \vdash^\varepsilon c_n \vdash^a d$ for some $n \in \mathbb{N}$.*

3 Proof Structure

The proof of the pumping lemma consists of three parts. In the first part we introduce a special kind of context free grammars (called well-formed grammars)

[3] In the following, we write push^1 whenever we mean some $\mathsf{push}^1_{\gamma,k}$ operation where the values of γ and k do not matter for the argument.

[4] In fact it is an edge-labelled graph; sets E_a need not to be disjoint.

for runs of a collapsible pushdown system $\mathcal{S}$. In such a grammar, each nonterminal represents a set of runs and each terminal is one of the transitions of $\mathcal{S}$. Let X and $X_1, \dots, X_m$ be sets of runs and δ some transition. A rule $X \supseteq \delta X_1 X_2 \dots X_m$ describes a run R if $R = S \circ T_1 \circ T_2 \circ \dots \circ T_n$ such that S performs only the transition δ and $T_i \in X_i$. A grammar describes a family $\mathcal{X}$ of sets of runs if the rules for each $X \in \mathcal{X}$ describe exactly the runs in X. Well-formed grammars are syntactically restricted in order to obtain the following result. If $\mathcal{X}$ is a finite family described by a well-formed grammar, we can define

1. a function $\mathsf{ctype}_{\mathcal{X}}$ from configurations of $\mathcal{S}$ to a finite partial order $(\mathcal{T}_{\mathcal{S}}, \sqsubseteq)$ (of *types of configurations*), and
2. for each $X \in \mathcal{X}$ a level $\mathsf{lev}(X) \in \{0, 1, \dots, n\}$

such that the following transfer property of runs holds.

Theorem 3.1. *Let $\mathcal{X}$ be a family of sets of runs described by a well-formed grammar, $R \in X \in \mathcal{X}$, and c be a configuration with $\mathsf{ctype}_{\mathcal{X}}(R(0)) \sqsubseteq \mathsf{ctype}_{\mathcal{X}}(c)$.*

1. *There is a run $S \in X$ starting in c which has the same final state as R and*
2. *if $\mathsf{lev}(X) = 0$, then $\mathsf{ctype}_{\mathcal{X}}(R(|R|)) \sqsubseteq \mathsf{ctype}_{\mathcal{X}}(S(|S|))$.*

The idea behind the definition of $\mathsf{ctype}_{\mathcal{X}}$ is that we assign a type not only to the whole configuration, but also to every k-stack (for every k). This type summarises possible behaviours of the k-stack in dependence on the type of the n-stack below this k-stack. This makes types compositive: the type of a stack $s^{k+1} : s^k$ is determined by the type of s^{k+1} and of s^k. The above theorem generalises results of [16] in two ways: first, it works for collapsible systems; second, it works for arbitrary well-formed grammars instead of a fixed family of sets of runs. The corresponding part of the proof from [16] is not transferable to collapsible systems at all. For collapsible systems we even need a new definition of types. We stress that the new definition of types relies on the different form of representing links in stacks: our k-stack already contains all linked stacks, so we can summarise it using a type from a finite set. On the other hand the original k-stack has arbitrarily many numbers pointing to stacks "outside", and we could not define a type from a finite set because the behaviour of a k-stack would depend on this unbounded context "outside".

In the second part of the proof (cf. Section 5), we introduce a well-formed grammar for a certain family $\mathcal{X}$. As a main feature, $\mathcal{X}$ contains the set of so-called *pumping runs* $\mathcal{P}$. In the grammar describing $\mathcal{X}$, the level of $\mathcal{P}$ is 0 whence the strong version of Theorem 3.1 applies. If a pumping run R starts and ends in configurations of the same type, this theorem then allows to pump this run, i.e., basically we can append a copy of this run to its end and iterating this process we obtain infinitely many pumping runs.

The last part of the proof uses Theorem 3.1 for the above family $\mathcal{X}$ to deduce the pumping lemma. This part follows closely the analogous proof for the non-collapsible pushdown systems in [16] (details in [10]): we prove that a long run contains a pumping run such that the application of Theorem 3.1 yields a configuration c on this path such that either the graph is infinitely branching at

c or the pumped runs yield longer and longer paths in the ε-contraction of the pushdown graph.

4 Run Grammars

Let $\mathcal{X}$ be a finite family whose elements are sets of runs of $\mathcal{S}$. We want to describe this family using a kind of context free grammar. In this grammar the members of $\mathcal{X}$ appear as nonterminals and the transitions of $\mathcal{S}$ play the role of terminals.

We assume that there is a partition $\mathcal{X} = \bigcup_{i=0}^{n} \mathcal{X}_i$ into pairwise distinct families of sets of runs. For each set $X \in \mathcal{X}$, we define its level to be $\mathsf{lev}(X) := i$ if $X \in \mathcal{X}_i$. We only consider *well-formed grammars* that satisfy the restriction that all rules of the grammar have to be *well-formed*.

Definition 4.1. *A* well-formed rule over $\mathcal{X}$ *(*wf-rule *for short) is of the form*

1. $X \supseteq$ *where* $X \in \mathcal{X}$*, or*
2. $X \supseteq \delta$ *where* $\delta \in \Delta$*,* $X \in \mathcal{X}$ *and if the operation in* δ *is* pop^k *or* col^k *then* $k \leq \mathsf{lev}(X)$*, or*
3. $X \supseteq \delta Y$ *where* $\delta \in \Delta$*,* $X, Y \in \mathcal{X}$*,* $\mathsf{lev}(Y) \leq \mathsf{lev}(X)$ *and if the operation in* δ *is* pop^k *or* col^k *then* $k \leq \mathsf{lev}(Y)$*, or*
4. $X \supseteq \delta YZ$ *where* $\delta \in \Delta$*,* $X, Y, Z \in \mathcal{X}$*,* $\mathsf{lev}(Z) \leq \mathsf{lev}(X)$*, if the operation in* δ *is* pop^k *or* col^k *then* $k \leq \mathsf{lev}(Y)$*, and whenever* R *is a composition of a one-step run performing transition* δ *with a run from* Y*, then the topmost* $\mathsf{lev}(Y)$*-stacks of* $R(0)$ *and* $R(|R|)$ *coincide.*

Definition 4.2. *We say that a run* R *is* described *by a wf-rule* $X \supseteq \delta X_1 \dots X_m$*,* $m \in \{0, 1, 2\}$ *if there is a decomposition* $R = R_0 \circ R_1 \circ \dots \circ R_m$ *such that* R_0 *has length* 1 *and performs* δ *and* $R_i \in X_i$ *for each* $1 \leq i \leq m$*; a run* R *is described by* $X \supseteq$ *if* $|R| = 0$*. We say that a family* $\mathcal{X}$ *is* described *by a well-formed grammar* $\mathcal{R}_{\mathcal{X}}$ *if for each* $X \in \mathcal{X}$*, a run* R *is in* X *if and only if it is described by some rule* $X \supseteq \delta X_1 \dots X_m \in \mathcal{R}_{\mathcal{X}}$*.*

Example 4.3. Let $\mathcal{Q}$ be the set of all runs. Setting $\mathsf{lev}(\mathcal{Q}) = n$, the one-element family $\{\mathcal{Q}\}$ is described by the wf-rules $\mathcal{Q} \supseteq \delta\mathcal{Q}$ for each transition δ, and $\mathcal{Q} \supseteq$.

Indeed, for every run R either $|R| = 0$ or R consists of a first transition followed by some run. Note that we cannot choose $\mathsf{lev}(\mathcal{Q})$ different from n whenever $\mathcal{S}$ contains a transition δ_0 performing col^n or pop^n. If we set $\mathsf{lev}(\mathcal{Q}) < n$, then $\mathcal{Q} \supseteq \delta_0\mathcal{Q}$ would not be a wf-rule.

Next we prove that the class of families described by well-formed grammars is closed under addition of unions and compositions. This is crucial for the decidability results mentioned in Lemma 1.3. If X and Y are sets of runs, we set $X \circ Y := \{R \circ S : R \in X, S \in Y\}$.

Lemma 4.4. *Let* $\mathcal{X}$ *be a family described by a well-formed grammar. For* $X, Y \in \mathcal{X}$ *the family* $\mathcal{X} \cup \{X \cup Y\}$ *is described by a well-formed grammar. Moreover, there is a family* $\mathcal{Y} \supseteq \mathcal{X} \cup \{X \circ Y\}$ *that is described by a well-formed grammar. In these grammars, we have* $\mathsf{lev}(X \cup Y) = \mathsf{lev}(X \circ Y) = \max(\mathsf{lev}(X), \mathsf{lev}(Y))$*.*

Proof. For each rule $Z \supseteq \delta Z_1 \dots Z_m$ with $Z \in \{X, Y\}$ adding the rule $(X \cup Y) \supseteq \delta Z_1 \dots Z_m$ settles the case of unions.

For the composition, we add a set $Z \circ Y$ for each $Z \in \mathcal{X}$, and a new set Y^i for $0 \leq i \leq n$. Y^i contains exactly the same runs as Y, but we set $\mathsf{lev}(Y^i) := \max(\mathsf{lev}(Y), i)$. Wf-rules describing Y^i are clearly obtained from the rules for Y by replacing the left-hand side by Y^i. Note that increasing the level of the left-hand size turns well-formed rules into well-formed rules. Rules for each of the $Z \circ Y$ are easily obtained from rules for Z as follows.

- If there is a rule $Z \supseteq$, for each rule having Y on the left side we add the same rule with $Z \circ Y$ on the left side,
- for each rule $Z \supseteq \delta$ we add a rule $(Z \circ Y) \supseteq \delta Y^{\mathsf{lev}(Z)}$,
- for each rule $Z \supseteq \delta X_1$ we add a rule $(Z \circ Y) \supseteq \delta(X_1 \circ Y)$,
- for each rule $Z \supseteq \delta X_1 X_2$ we add a rule $(Z \circ Y) \supseteq \delta X_1(X_2 \circ Y)$.

It is straightforward to check that this is a well-formed grammar describing the family $\mathcal{Y} := \mathcal{X} \cup \{Z \circ Y : Z \in \mathcal{X}\} \cup \{Y^i : 0 \leq i \leq n\}$. □

5 A Family of Runs

We now define a family $\mathcal{X}$ described by a well-formed grammar. We first name the sets of runs that we define in the following. Some of our classes have subscripts from ε, $\not\varepsilon$, $=$, and $<$. Subscript ε marks a set if all runs in the set only perform ε-transitions, while $\not\varepsilon$ marks a set if each run in the set performs at least one non-ε-transitions. Subscript $<$ marks sets (of pumping runs) where each run starts in a smaller stack than it ends, while $=$ marks sets where no run starts in a smaller stack than it ends (it follows that each such pumping run ends in the same stack as it starts). $\mathcal{X}$ consists of the following sets (which we describe in detail on the following pages).

- $\mathcal{Q}$ of all runs,
- $\mathcal{N}_k$ of top^k-non-erasing runs,
- $\mathcal{P}$ of pumping runs which is the disjoint union of the sets $\mathcal{P}_{x,y}$ for $x \in \{<, =\}$, $y \in \{\varepsilon, \not\varepsilon\}$. Additionally, we set $\mathcal{P}_{\not\varepsilon} = \mathcal{P}_{<,\not\varepsilon} \cup \mathcal{P}_{=,\not\varepsilon}$ and $\mathcal{P}_{\varepsilon} = \mathcal{P}_{<,\varepsilon} \cup \mathcal{P}_{=,\varepsilon}$.
- $\mathcal{R}_{k,j}$ of k-returns of change level $j \geq k$ which is the disjoint union of the sets $\mathcal{R}_{k,j,y}$ for $y \in \{\varepsilon, \not\varepsilon\}$, and
- $\mathcal{C}_{k,j}$ of k-colreturns of change level $j \geq k$ which is the disjoint union of the sets $\mathcal{C}_{k,j,y}$ for $y \in \{\varepsilon, \not\varepsilon\}$.

In order to easily distinguish between ε-runs and $\not\varepsilon$-runs in the rules, we partition the transition relation $\Delta = \Delta_\varepsilon \cup \Delta_{\not\varepsilon}$ such that Δ_ε contains exactly the ε-labelled transitions. Before we can give rules for the family we need to define the levels of its sets. We set $\mathsf{lev}(\mathcal{Q}) = n, \mathsf{lev}(\mathcal{R}_{k,j,y}) = k, \mathsf{lev}(\mathcal{C}_{k,j,y}) = k, \mathsf{lev}(\mathcal{N}_k) = n$ and $\mathsf{lev}(\mathcal{P}_{x,y}) = 0$.

Now we give rules for these sets and we describe the main properties of runs in each of the sets.

Recall that we have described $\mathcal{Q}$ by well-formed rules in Example 4.3. The sets of returns and colreturns are auxiliary sets. Returns occur in the wf-rules for $\mathcal{N}_k$ and $\mathcal{P}_{x,y}$ while colreturns are necessary to give wf-rules for returns.

$\mathcal{N}_k$ contains all runs R where the topmost k-stack of $R(0)$ is never removed during the run. First, we give an idea how the set $\mathcal{N}_0$ plays an important role in our pumping lemma. Recall that we want to apply Theorem 3.1 to pumping runs in order to obtain arbitrarily many runs starting in a given configuration. Our final goal is to construct infinitely many different paths in the ε-contraction of the graph of a given collapsible pushdown system that all end in a specific state q. But in general, the pumping runs we construct end in a different state. Thus, the type of the stack reached by each of the pumping runs should determine that we can reach a configuration with state q from this position. This could be done using the set $\mathcal{Q}$ but it is not enough: if the pumping runs induce ε-labelled paths then we could append runs from $\mathcal{Q}$ that all lead to the same configuration. In this case, we construct longer and longer runs but all these runs encode the same edge in the ε-contraction. This is prohibited by the use of runs from $\mathcal{N}_0$: we can prove that the longer pumping runs we construct end in larger stacks. Appending a run from $\mathcal{N}_0$ to such a run ensures that the resulting run also ends in a large stack. From this observation we will obtain infinitely many runs that end in different configurations with state q. Thus, they induce infinitely many paths in the ε-contraction. The rules for $\mathcal{N}_k$ are

– $\mathcal{N}_k \supseteq$,
– $\mathcal{N}_k \supseteq \delta\mathcal{N}_k$ for each $\delta \in \Delta$ performing an operation of level at most k,
– $\mathcal{N}_k \supseteq \delta^j\mathcal{N}_{j-1}$ for each $\delta^j \in \Delta$ performing a push^j and $j \geq k+1$,
– $\mathcal{N}_k \supseteq \delta^j\mathcal{R}_{j,j}\mathcal{N}_k$ for each $\delta^j \in \Delta$ performing a push^j.

Our analysis of returns reveals that δ^j followed by a run from $\mathcal{R}_{j,j}$ starts and ends in the same stack. Thus, the last rule satisfies the requirement that the topmost j-stacks of these stacks coincide. Moreover, such a run never changes the topmost j-stack of the initial configuration. Using this fact it is straightforward to see that every run described by these rules does not remove the topmost k-stack. The other direction is more involved and the proof can be found in [10].

Some run R is a pumping run, i.e., $R \in \mathcal{P}$, if its final stack is created completely on top of its initial stack in the following sense: the topmost 1-stack of $R(|R|)$ is obtained as a (possibly modified) copy of the topmost 1-stack of $R(0)$, and in this copy the topmost 0-stack of $R(0)$ was never removed. Another view on this definition is as follows: for each k, the run R may look into a copy of the topmost k-stack of $R(0)$ only if this copy is not directly involved in the creation of the topmost k-stack of $R(|R|)$. In [10], we define a history function that makes the notion of being involved in the creation of some stack precise: for each $i < |R|$ and for each k-stack s^k of $R(|R|)$ we can identify a k-stack t^k in $R(i)$ which is the maximal k-stack involved in the creation of this stack.

In the rest of this section y, y_0, y_1, y_2 are variables in $\{\varepsilon, \not\varepsilon\}$ where we assume that either all are ε or $y = \not\varepsilon$ and one of the y_i occurring in the rule is $\not\varepsilon$. j is a variable ranging over $\{1, 2, \ldots, n\}$. The rules for $\mathcal{P}$ are

- $\mathcal{P}_{=,\varepsilon} \supseteq$,
- $\mathcal{P}_{<,y} \supseteq \delta_{y_0}\mathcal{P}_{x,y_1}$ for each $\delta_{y_0} \in \Delta_{y_0}$ performing push^j and $x \in \{=,<\}$,
- $\mathcal{P}_{x,y} \supseteq \delta^j_{y_0}\mathcal{R}_{j,j,y_1}\mathcal{P}_{x,y}$ for each $\delta^j_{y_0} \in \Delta_{y_0}$ performing push^j and $x \in \{=,<\}$,
- $\mathcal{P}_{<,y} \supseteq \delta^j_{y_0}\mathcal{R}_{j,j',y_1}\mathcal{P}_{x,y_2}$ for each $\delta^j_{y_0} \in \Delta_{y_0}$ performing push^j, $j' > j$ and $x \in \{=,<\}$.

Proving the correctness of this set of rules with respect to our intended meaning of the sets $\mathcal{P}_{x,y}$ requires a detailed study of returns which can be found in [10]. In order to see that the rules of the last two kinds are well-formed we need the property that for every run R which first performs a push^j operation followed by a j-return, the topmost j-stack of $R(0)$ and $R(|R|)$ is the same.

Example 5.1. A run of length 1 performing a push^1 operation is a pumping run. Also a run of length 2 performing a push^1 operation followed by a pop^1 operation is a pumping run. However a run of length 2 performing first a pop^1 operation and then a push^1 operation is not a pumping run. This shows that in the definition of a pumping run we do not only care about the initial and final configuration, but about the way the final configuration is created by the run: a pumping run R may never remove the topmost 0-stack of $R(0)$.

Next consider a run R of length 3 performing the sequence of operations push^2, pop^1, pop^2. It is also a pumping run. Notice that this run "looks" into a copy of the topmost 1-stack of $R(0)$, i.e., it removes its topmost 0-stack whence it depends on symbols of $R(0)$ other than the topmost one. One can see that in any 2-CPS, whenever a pumping run R looks into a copy of the topmost 1-stack of $R(0)$, then this copy is completely removed from the stack at some later point in the run. However, this is not true for higher levels. A counter example is a run performing push^2, pop^1, push^3, pop^2.

Next we define returns. A run R is a k-return (where $1 \leq k \leq n$) if

- the topmost $(k-1)$-stack of $R(|R|)$ is obtained as a copy of the second topmost $(k-1)$-stack of $R(0)$ (in particular we require that there are at least two $(k-1)$-stacks in the topmost k-stack of $R(0)$), and
- while tracing this copy of the second topmost $(k-1)$-stack of $R(0)$ which finally becomes the topmost $(k-1)$-stack of $R(|R|)$, it is not the topmost $(k-1)$-stack of $R(i)$ for any $i < |R|$.

Additionally, for a k-return R its change level is the maximal j such that the topmost j-stack of the initial and of the final stack of R differ in size (i.e. in the number of $(j-1)$-stacks they contain).[5] One can see that the topmost k-stack of $R(0)$ is always greater by one than the topmost k-stack of $R(|R|)$, so we have $j \geq k$. Recall that $\mathcal{R}_{k,j}$ is the set of k-returns of change level j.

Let us just give some intuition about returns before we state their exact characterisation using wf-rules. The easiest sets of returns are those where $k = j$. A run $R \in \mathcal{R}_{k,k}$ starts in some stack s, ends in the stack $\mathsf{pop}^k(s)$, and never visits

[5] One can see that it is the same as saying that the topmost j-stack of the initial and of the final stack of R differ. However a definition using size is more convenient.

$\mathsf{pop}^k(s)$ (or any smaller stack) before the final configuration. Notice also that there is a minor restriction on the use of collapse operations: R is not allowed to use links of level k stored in s in order to reach $\mathsf{pop}^k(s)$. Indeed, if such a link was used, then the topmost $(k-1)$-stack of $R(|R|)$ would not be a copy of the second topmost $(k-1)$-stack of $R(0)$, but a copy of the $(k-1)$-stack stored in the link used. Note that this distinction is due to our special representation of links, yet it is useful for the understanding of the definitions.

In the case that $j > k$ things are more complicated but similar. This time $R \in \mathcal{R}_{k,j}$ makes a number of copies of the (possibly modified) topmost $(j-1)$-stack of the initial stack s whence the topmost j-stack of the final stack s' is of bigger size than the topmost j-stack of s. But again the topmost k-stack of s' is the same as the topmost k-stack of $\mathsf{pop}^k(s)$, and is in fact created as a modified copy of the topmost k-stack of s. Furthermore, while tracing the history of this copy along the configurations of the run, the size of this copy is always greater than its size in $R(|R|)$. Notice however that we may also create some other copies of the topmost k-stack of s, in which we can remove arbitrarily many $(k-1)$-stacks. Finally, there is again a minor restriction on the use of collapse links stored in the initial stack s. This restriction implies that the stack obtained via application of the stack operations of the return to s is independent of the linked stacks, i.e., if we replace one of the links of the stack s such that the stack operations of R are still applicable to the resulting stack s', then this sequence of stack operations applied to s' results in the same stack (as when applied to s). In Example 5.4 we discuss the conditions under which we may use a link stored in the initial stack of some return.

Example 5.2. Consider a run R of length 6 which performs the sequence

$$\mathsf{push}^2, \mathsf{pop}^1, \mathsf{pop}^2, \mathsf{pop}^1, \mathsf{push}^1, \mathsf{pop}^1.$$

Below we use the notation that symbols taken in square brackets are in one 1-stack (omitting collapse links). Started in $[aa][aa]$, the configurations of R are

$$R(0) = [aa][aa][aa],\ R(1) = [aa][aa][a],\ R(2) = [aa][aa],$$
$$R(3) = [aa][a],\ R(4) = [aa][aa],\ R(5) = [aa][a].$$

We have $R{\restriction}_{0,2} \in \mathcal{R}_{1,2}$, $R{\restriction}_{0,4}, R{\restriction}_{1,2}, R{\restriction}_{3,4}, R{\restriction}_{5,6} \in \mathcal{R}_{1,1}$ and $R{\restriction}_{1,3}, R{\restriction}_{2,3} \in \mathcal{R}_{2,2}$. These are the only subruns of R being returns, in particular R is not a 1-return because it visits its final stack before its final configuration.

Example 5.3. The run R of length 5 performing the sequence of operations

$$\mathsf{push}^2, \mathsf{pop}^1, \mathsf{push}^3, \mathsf{pop}^2, \mathsf{pop}^1$$

is a 1-return of change level 3. Notice that the final stack contains a copy of the topmost 1-stack of $R(0)$ with its topmost 0-stack removed.

The rules for returns are as follows:

- $\mathcal{R}_{k,k,y} \supseteq \delta_{y_0}$ for each $\delta_{y_0} \in \Delta_{y_0}$ performing pop^k,
- $\mathcal{R}_{k,j,y} \supseteq \delta_{y_0}\mathcal{R}_{k,j,y_1}$ for each $\delta_{y_0} \in \Delta_{y_0}$ performing an operation of level $< k$,
- $\mathcal{R}_{k,j,y} \supseteq \delta_{y_0}^{j_0}\mathcal{R}_{k,j_1,y_1}$ for each $\delta_{y_0}^{j_0} \in \Delta_{y_0}$ performing a push^{j_0} such that $j_0 > k$ and $\max\{j_0, j_1\} = j$,
- $\mathcal{R}_{k,j,y} \supseteq \delta_{y_0}^{j_0}\mathcal{R}_{j_0,j_0,y_1}\mathcal{R}_{k,j,y_2}$ for each $\delta_{y_0}^{j_0} \in \Delta_{y_0}$ performing a push of level j_0,
- $\mathcal{R}_{k,j,y} \supseteq \delta_{y_0}^{j_0}\mathcal{R}_{j_0,j_1,y_1}\mathcal{R}_{k,j_2,y_2}$ for each $\delta_{y_0}^{j_0} \in \Delta_{y_0}$ performing a push^{j_0} such that $j_1 > j_0$ and $\max\{j_1, j_2\} = j$, and
- $\mathcal{R}_{k,j,y} \supseteq \delta_{y_0}\mathcal{C}_{k,j,y_1}$ for each $\delta_{y_0} \in \Delta_{y_0}$ performing a $\mathsf{push}^1_{a,k}$.

A k-colreturn is a run R that performs in the last step a col^k on a copy of the topmost symbol of its initial stack. The change level of k-colreturns is (again) defined as the maximal j such that the topmost j-stack of the initial and of the final stack of the colreturn R differ in size.

Note that k-colreturns appear in the rules for returns after a push of level 1. The simplest example of a return described by the last rule is a run R starting in a stack s and performing $\mathsf{push}^1_{a,k}$ and then col^k. Note that such a sequence has the same effect as applying pop^k to s. Note that $R{\restriction}_{1,2}$ in this example is a run from the stack $s' := \mathsf{push}^1_{a,k}(s)$ to $\mathsf{pop}^k(s')$ (for $k \geq 2$). Nevertheless we exclude it from the definition of a k-return of change level k because this effect is not transferable to arbitrary other stacks: of course, we can apply the transition of $R{\restriction}_{1,2}$ to the stack $\mathsf{push}^k(s')$ and obtain a run R' from $\mathsf{push}^k(s')$ to $\mathsf{pop}^k(s')$. But apparently this is not a run from some stack s'' to a stack $\mathsf{pop}^k(s'')$, so it is not a k-return. For this reason our definition of returns disallows the application of certain stored collapse links. The colreturns take care of such situations where we use the links stored in the stack. Notice that k-colreturns occur in the rules defining the other sets of runs only at those points where we performed a push of level 1 whence we can be sure that the effect of the collapse operation coincides with the application of exactly one pop^k operation to the initial stack.

Example 5.4. Consider a run R of length 4 performing $\mathsf{push}^2, \mathsf{col}^1, \mathsf{pop}^2, \mathsf{pop}^1$. It is a 1-return of change level 1. Notice that it performs a collapse operation using a (copy of a) link already stored in $R(0)$. But $R{\restriction}_{1,3}$ is a 2-return (of change level 2) which covers this collapse operation, i.e., whenever the whole sequence is applicable to some stack s it ends in the stack $\mathsf{pop}^1(s)$. As a general rule, we allow the use of a col^k from a (copy) of a link stored in the initial stack of some return R if it occurs within some subrun R' that is a k'-return or k'-colreturn of higher level(i.e., $k' > k$). In such cases the resulting stack does not depend on the stack stored in the link (as long as the whole sequence of operations of the return is applicable). Hence, the following sequence of operations also induces a 1-return of change level 1: $\mathsf{push}^3, \mathsf{col}^2, \mathsf{pop}^3, \mathsf{pop}^1$.

Finally, let us state the rules for k-colreturns.

- $\mathcal{C}_{k,k,y} \supseteq \delta_{y_0}$ for each $\delta_{y_0} \in \Delta_{y_0}$ performing a col^k,

- $\mathcal{C}_{k,j,y} \supseteq \delta^{j_0}_{y_0}\mathcal{C}_{k,j_1,y_1}$ for each $\delta^{j_0}_{y_0} \in \Delta_{y_0}$ performing a push^{j_0} such that $j_0 \geq 2$ and $\max\{j_0, j_1\} = j$,
- $\mathcal{C}_{k,j,y} \supseteq \delta^{j_0}_{y_0}\mathcal{R}_{j_0,j_0,y_1}\mathcal{C}_{k,j,y_2}$ for each $\delta^{j_0}_{y_0} \in \Delta_{y_0}$ performing a push^{j_0}, and
- $\mathcal{C}_{k,j,y} \supseteq \delta^{j_0}_{y_0}\mathcal{R}_{j_0,j_1,y_1}\mathcal{C}_{k,j_2,y_2}$ for each $\delta^{j_0}_{y_0} \in \Delta_{y_0}$ performing a push^{j_0} such that $j_1 > j_0$ and $\max\{j_1, j_2\} = j$.

This completes the presentation of the well-formed grammar describing $\mathcal{X}$.

References

1. Aehlig, K., de Miranda, J.G., Ong, C.-H.L.: Safety Is not a Restriction at Level 2 for String Languages. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 490–504. Springer, Heidelberg (2005)
2. Blumensath, A.: On the structure of graphs in the caucal hierarchy. Theor. Comput. Sci. 400(1-3), 19–45 (2008)
3. Broadbent, C., Carayol, A., Hague, M., Serre, O.: A Saturation Method for Collapsible Pushdown Systems. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) Automata, Languages, and Programming, Part II. LNCS, vol. 7392, pp. 165–176. Springer, Heidelberg (2012)
4. Carayol, A., Wöhrle, S.: The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 112–123. Springer, Heidelberg (2003)
5. Caucal, D.: On Infinite Terms Having a Decidable Monadic Theory. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 165–176. Springer, Heidelberg (2002)
6. Gilman, R.H.: A shrinking lemma for indexed languages. Theor. Comput. Sci. 163(1&2), 277–281 (1996)
7. Hague, M., Murawski, A.S., Ong, C.-H.L., Serre, O.: Collapsible pushdown automata and recursion schemes. In: LICS, pp. 452–461. IEEE Computer Society (2008)
8. Hayashi, T.: On derivation trees of indexed grammars. Publ. RIMS, Kyoto Univ. 9, 61–92 (1973)
9. Kartzow, A.: A pumping lemma for collapsible pushdown graphs of level 2. In: Bezem, M. (ed.) CSL. LIPIcs, vol. 12, pp. 322–336. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)
10. Kartzow, A., Parys, P.: Strictness of the collapsible pushdown hierarchy. CoRR, abs/1201.3250 (2012)
11. Knapik, T., Niwiński, D., Urzyczyn, P.: Higher-Order Pushdown Trees Are Easy. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 205–222. Springer, Heidelberg (2002)
12. Maslov, A.N.: The hierarchy of indexed languages of an arbitrary level. Soviet Math. Dokl. 15, 1170–1174 (1974)
13. Maslov, A.N.: Multilevel stack automata. Problems of Information Transmission 12, 38–43 (1976)
14. Parys, P.: Collapse operation increases expressive power of deterministic higher order pushdown automata. In: Schwentick, T., Dürr, C. (eds.) STACS. LIPIcs, vol. 9, pp. 603–614. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)
15. Parys, P.: On the significance of the collapse operation. In: To appear in LICS (2012)
16. Parys, P.: A pumping lemma for pushdown graphs of any level. In: Dürr, C., Wilke, T. (eds.) STACS. LIPIcs, vol. 14, pp. 54–65. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)

Computational Complexity of Smooth Differential Equations

Akitoshi Kawamura[1], Hiroyuki Ota[1], Carsten Rösnick[2], and Martin Ziegler[2]

[1] University of Tokyo, Tokyo, Japan
[2] Technische Universität Darmstadt, Darmstadt, Germany

Abstract. The computational complexity of the solution h to the ordinary differential equation $h(0) = 0$, $h'(t) = g(t, h(t))$ under various assumptions on the function g has been investigated in hope of understanding the intrinsic hardness of solving the equation numerically. Kawamura showed in 2010 that the solution h can be PSPACE-hard even if g is assumed to be Lipschitz continuous and polynomial-time computable. We place further requirements on the smoothness of g and obtain the following results: the solution h can still be PSPACE-hard if g is assumed to be of class C^1; for each $k \geq 2$, the solution h can be hard for the counting hierarchy if g is of class C^k.

1 Introduction

Let $g\colon [0,1] \times \mathbf{R} \to \mathbf{R}$ be continuous and consider the differential equation

$$h(0) = 0 \ , \qquad\qquad Dh(t) = g(t, h(t)) \quad t \in [0,1] \ , \tag{1}$$

where Dh denotes the derivative of h. How complex can the solution h be, assuming that g is polynomial-time computable? Here, polynomial-time computability and other notions of complexity are from the field of *Computable Analysis* [1,2] and measure how hard it is to approximate real functions with specified precision (Sect. 2).

If we put no assumption on g other than being polynomial-time computable, the solution h (which is not unique in general) can be non-computable. Table 1 summarizes known results about the complexity of h under various assumptions (that get stronger as we go down the table). In particular, if g is (globally) Lipschitz continuous, then the (unique) solution h is known to be polynomial-space computable but still can be PSPACE-hard [3]. In this paper, we study the complexity of h when we put stronger assumptions about the smoothness of g.

In numerical analysis, knowledge about smoothness of the input function (such as being differentiable enough times) is often beneficial in applying certain algorithms or simplifying their analysis. However, to our knowledge, this casual understanding that smoothness is good has not been rigorously substantiated in terms of computational complexity theory. This motivates us to ask whether, for our differential equation (1), smoothness really reduces the complexity of the solution.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 578–589, 2012.

Table 1. The complexity of the solution h of (1) assuming g is polynomial-time computable

Assumptions	Upper bounds	Lower bounds
—	—	can be all non-computable [4]
h is the unique solution	computable [5]	can take arbitrarily long time [6,7]
the Lipschitz condition	polynomial-space [6]	can be PSPACE-hard [3]
g is of class $\mathrm{C}^{(\infty,1)}$	polynomial-space	can be PSPACE-hard (Theorem 1)
g is of class $\mathrm{C}^{(\infty,k)}$ (for each constant k)	polynomial-space	can be CH-hard (Theorem 2)
g is analytic	polynomial-time [8,9,10]	—

One extreme is the case where g is analytic: h is then polynomial-time computable (the last row of the table) by an argument based on Taylor series[1] (this does not necessarily mean that computing the values of h from those of g is easy; see the last paragraph of Sect. 4). Thus our interest is in the cases between Lipschitz and analytic (the fourth and fifth rows). We say that g is of class $\mathrm{C}^{(i,j)}$ if the partial derivative $D^{(n,m)}g$ (often also denoted $\partial^{n+m}g(t,y)/\partial t^n \partial y^m$) exists and is continuous for all $n \leq i$ and $m \leq j$;[2] it is said to be of class $\mathrm{C}^{(\infty,j)}$ if it is of class $\mathrm{C}^{(i,j)}$ for all $i \in \mathbf{N}$.

Theorem 1. *There exists a polynomial-time computable function $g\colon [0,1] \times [-1,1] \to \mathbf{R}$ of class $\mathrm{C}^{(\infty,1)}$ such that the equation (1) has a* PSPACE*-hard solution $h\colon [0,1] \to \mathbf{R}$.*

Theorem 2. *Let k be a positive integer. There is a polynomial-time computable function $g\colon [0,1] \times [-1,1] \to \mathbf{R}$ of class $\mathrm{C}^{(\infty,k)}$ such that the equation (1) has a* CH*-hard solution $h\colon [0,1] \to \mathbf{R}$, where* CH $\subseteq$ PSPACE *is the Counting Hierarchy (see Sect. 3.2).*

We said $g\colon [0,1] \times [-1,1] \to \mathbf{R}$ instead of $g\colon [0,1] \times \mathbf{R} \to \mathbf{R}$, because the notion of polynomial-time computability of real functions in this paper is defined only

[1] As shown by Müller [8] and Ko and Friedman [9], polynomial-time computability of an analytic function on a compact interval is equivalent to that of its Taylor sequence at a point (although the latter is a local property, polynomial-time computability on the whole interval is implied by analytic continuation; see [8, Corollary 4.5] or [10, Theorem 11]). This implies the polynomial-time computability of h, since we can efficiently compute the Taylor sequence of h from that of g.

[2] Another common terminology is to say that g is of class C^k if it is of class $\mathrm{C}^{(i,j)}$ for all i, j with $i + j \leq k$.

when the domain is a bounded closed region.[3] This makes the equation (1) ill-defined in case h ever takes a value outside $[-1, 1]$. By saying that h is a solution in Theorem 1, we are also claiming that $h(t) \in [-1, 1]$ for all $t \in [0, 1]$. In any case, since we are putting stronger assumptions on g than Lipschitz continuity, such a solution h, if it exists, is unique.

Whether smoothness of the input function reduces the complexity of the output has been studied for operators other than solving differential equations, and the following negative results are known. The integral of a polynomial-time computable real function can be #P-hard, and this does not change by restricting the input to C^∞ (infinitely differentiable) functions [1, Theorem 5.33]. Similarly, the function obtained by maximization from a polynomial-time computable real function can be NP-hard, and this is still so even if the input function is restricted to C^∞ [1, Theorem 3.7][4]. (Restricting to analytic inputs renders the output polynomial-time computable, again by the argument based on Taylor series.) In contrast, for the differential equation we only have Theorem 2 for each k, and do not have any hardness result when g is assumed to be infinitely differentiable.

Theorems 1 and 2 are about the complexity of each solution h. We can also talk about the complexity of the operator that maps g to h; see Sect. 4.

Notation. Let $\mathbf{N}$, $\mathbf{Z}$, $\mathbf{Q}$, $\mathbf{R}$ denote the set of natural numbers, integers, rational numbers and real numbers, respectively.

Let A and B be bounded closed intervals in $\mathbf{R}$. We write $|f| = \sup_{x \in A} f(x)$ for $f \colon A \to \mathbf{R}$. A function $f \colon A \to \mathbf{R}$ is *of class* C^i (i-times continuously differentiable) if all the derivatives $Df, D^2 f, \dots, D^i f$ exist and are continuous.

For a differentiable function g of two variables, we write $D_1 g$ and $D_2 g$ for the derivatives of g with respect to the first and the second variable, respectively. A function $g \colon A \times B \to \mathbf{R}$ is of *class* $\mathrm{C}^{(i,j)}$ if for each $n \in \{0, \dots, i\}$ and $m \in \{0, \dots j\}$, the derivative $D_1^n D_2^m g$ exists and is continuous. A function g is of *class* $\mathrm{C}^{(\infty,j)}$ if it is of class $\mathrm{C}^{(i,j)}$ for all $i \in \mathbf{N}$. When g is of class $\mathrm{C}^{(i,j)}$, we write $D^{(i,j)} g$ for the derivative $D_1^i D_2^j g$.

[3] Although we could extend our definition to functions with unbounded domain [11, Sect. 4.1], the results in Table 1 do not hold as they are, because polynomial-time compubable functions g, such as $g(t, y) = y + 1$, could yield functions h, such as $h(t) = \exp t - 1$, that grow too fast to be polynomial-time (or even polynomial-space) computable. Bournez, Graça and Pouly [12, Theorem 2] report that the statement about the analytic case holds true if we restrict the growth of h (and its extention to the complex plane) appropriately.

[4] The proof of this fact in [1, Theorem 3.7] needs to be fixed by redefining

$$f(x) = \begin{cases} u_s & \text{if not } R(s,t), \\ u_s + 2^{-(p(n)+2n+1)\cdot n} \cdot h_1(2^{p(n)+2n+1}(x - y_{s,t})) & \text{if } R(s,t). \end{cases}$$

2 Computational Complexity of Real Functions

This section reviews the complexity notions in Computable Analysis [1,2]. We start by fixing an encoding of real numbers by string functions.

Definition 3. *A function $\phi\colon \{0\}^* \to \{0,1\}^*$ is a* name *of a real number x if for all $n \in \mathbf{N}$, $\phi(0^n)$ is the binary representation of $\lfloor x \cdot 2^n \rfloor$ or $\lceil x \cdot 2^n \rceil$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ mean rounding down and up to the nearest integer.*

In effect, a name of a real number x receives 0^n and returns an approximation of x with precision 2^{-n}.

We use *oracle Turing machines* (henceforth just *machines*) to work on these names (Fig. 1). Let M be a machine and ϕ be a function from strings to strings. We write $M^\phi(0^n)$ for the output string when M is given ϕ as oracle and string 0^n as input. Thus we also regard M^ϕ as a function from strings to strings.

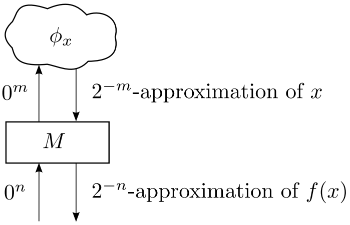

ϕ_x
0^m
2^{-m}-approximation of x
M
0^n
2^{-n}-approximation of $f(x)$

Fig. 1. A machine M computing a real function f

Definition 4. *Let A be a bounded closed interval of $\mathbf{R}$. A machine M* computes *a real function $f\colon A \to \mathbf{R}$ if for any $x \in A$ and any name ϕ_x of it, M^{ϕ_x} is a name of $f(x)$.*

Computation of a function $f\colon A \to \mathbf{R}$ on a two-dimensional bounded closed region $A \subseteq \mathbf{R}^2$ is defined in a similar way using machines with two oracles. A real function is (*polynomial-time*) *computable* if there exists some machine that computes it (in polynomial time). Polynomial-time computability of a real function f means that for any $n \in \mathbf{N}$, an approximation of $f(x)$ with error bound 2^{-n} is computable in time polynomial in n independent of the real number x.

By the time the machine outputs the approximation of $f(x)$ of precision 2^{-n}, it knows x only with some precision 2^{-m}. This implies that all computable real functions are continuous. If the machine runs in polynomial time, this m is bounded polynomially in n. Hence for every polynomial-time computable real function, there is a polynomial p such that $|f(x)-f(y)| \le 2^{-n}$ for all $x, y \in \mathrm{dom} f$ and $n \in \mathbf{N}$ with $|x-y| \le 2^{-p(n)}$.

To talk about hardness, we define reduction. A language $L \subseteq \{0,1\}^*$ is identified with the function $L\colon \{0,1\}^* \to \{0,1\}$ such that $L(u) = 1$ when $u \in L$.

Definition 5. *A language L* reduces *to a function $f: [0,1] \to \mathbf{R}$ if there exists a polynomial-time function S and a polynomial-time oracle Turing machine M (Fig. 2) such that for any string u,*

(i) *$S(u, \cdot)$ is a name of a real number x_u, and*
(ii) *$M^{\phi}(u)$ accepts if and only if $u \in L$ for any name ϕ of $f(x_u)$.*

This definition may look stronger than the one in Kawamura [3], but is easily seen to have the same power. For a complexity class C, a function f is *C-hard* if all languages in C reduce to f.

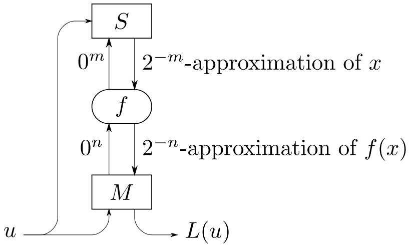

S
0^m
2^{-m}-approximation of x
f
0^n
2^{-n}-approximation of $f(x)$
M
u
$L(u)$

Fig. 2. Reduction from a language L to a function $f: [0,1] \to \mathbf{R}$

3 Proof of the Theorems

The proofs of Theorems 1 and 2 proceed as follows. In Sect. 3.1, we define *difference equations*, a discrete version of the differential equations. In Sect. 3.2, we show the PSPACE- and CH-hardness of difference equations with certain restrictions. In Sect. 3.3, we show that these classes of difference equations can be simulated by families of differential equations satisfying certain uniform bounds on higher-order derivatives. In Sect. 3.4, we prove the theorems by putting these families of functions together to obtain one differential equation having the desired smoothness ($\mathrm{C}^{(\infty,1)}$ and $\mathrm{C}^{(\infty,k)}$).

The idea of simulating a discrete system of limited feedback capability by differential equations was essentially already present in the proof of the Lipschitz version [3]. We look more closely at this limited feedback mechanism, and observe that this restriction is one on the *height* of the difference equation. We show that a stronger height restriction makes the difference equation simulable by smoother differential equations, leading to the CH-hardness for $\mathrm{C}^{(\infty,k)}$ functions.

3.1 Difference Equations

In this section, we define difference equations, a discrete version of differential equations, and show the PSPACE- and CH-hardness of families of difference equations with different height restrictions.

Let $[n]$ denote $\{0, \dots, n-1\}$. Let $G\colon [P] \times [Q] \times [R] \to \{-1, 0, 1\}$ and $H\colon [P+1] \times [Q+1] \to [R]$. We say that H is the solution of the *difference equation* given by G if for all $i \in [P]$ and $T \in [Q]$ (Fig. 3),

$$H(i, 0) = H(0, T) = 0 \ , \tag{2}$$

$$H(i+1, T+1) - H(i+1, T) = G(i, T, H(i, T)) \ . \tag{3}$$

We call P, Q and R the *height*, *width* and *cell size* of the difference equation. The equations (2) and (3) are similar to the initial condition $h(0) = 0$ and the equation $Dh(t) = g(t, h(t))$ in (1), respectively. In Sect. 3.3, we will simulate difference equations by differential equations using this similarity.

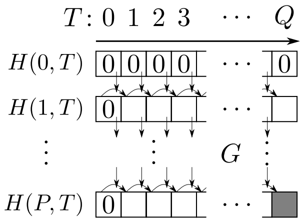


Fig. 3. The solution H of the difference equation given by G

We view a family of difference equations as a computing system by regarding the value of the bottom right cell (the gray cell in Fig. 3) as the output. A family $(G_u)_u$ of functions $G_u\colon [P_u] \times [Q_u] \times [R_u] \to \{-1, 0, 1\}$ *recognizes* a language L if for each u, the difference equation given by G_u has a solution H_u and $H_u(P_u, Q_u) = L(u)$. A family $(G_u)_u$ is *uniform* if the height, width and cell size of G_u are polynomial-time computable from u (in particular, they must be bounded by $2^{p(|u|)}$, for some polynomial p) and $G_u(i, T, Y)$ is polynomial-time computable from (u, i, T, Y). A family $(G_u)_u$ has *polynomial height* if the height P_u is bounded by some polynomial $p(|u|)$. A family $(G_u)_u$ has *logarithmic height* if the height P_u is bounded by $c \log |u| + d$ with some constants c and d. With this terminology, the key lemma in [3, Lemma 4.7] can be written as follows:

Lemma 6. *There exists a* PSPACE*-hard language L that is recognized by some uniform family of functions with polynomial height*[5]*.*

Kawamura obtained the hardness result in the third row in Table 1 by simulating the difference equations of Lemma 6 by Lipschitz-continuous differential equations. Likewise, Theorem 1 follows from Lemma 6, by a modified construction that keeps the function in class $\mathrm{C}^{(\infty,1)}$ (Sects. 3.3 and 3.4).

We show further that difference equations restricted to have logarithmic height can be simulated by $\mathrm{C}^{(\infty,k)}$ functions for each k (Sects. 3.3 and 3.4). Theorem 2 follows from this simulation and the following lemma.

[5] In fact, the languages recognized by uniform families with polynomial height coincide with PSPACE.

Lemma 7. *There exists a* CH*-hard language* L *that is recognized by some uniform family of functions with logarithmic height.*

The definition of the counting hierarchy CH, its connection to difference equations and the proof of Lemma 7 will be presented in Sect. 3.2.

3.2 The Counting Hierarchy and Difference Equations of Logarithmic Height

The polynomial hierarchy PH is defined using non-deterministic polynomial-time oracle Turing machines:

$$\Sigma_0^p = \mathsf{P} \ , \qquad \Sigma_{n+1}^p = \mathsf{NP}^{\Sigma_n^p} \ , \qquad \mathsf{PH} = \bigcup_n \Sigma_n^p \ . \tag{4}$$

The counting hierarchy CH is defined similarly using probabilistic polynomial-time oracle Turing machines [13,14]:

$$\mathsf{C}_0\mathsf{P} = \mathsf{P} \ , \qquad \mathsf{C}_{n+1}\mathsf{P} = \mathsf{PP}^{\mathsf{C}_n\mathsf{P}} \ , \qquad \mathsf{CH} = \bigcup_n \mathsf{C}_n\mathsf{P} \ . \tag{5}$$

It is known that $\mathsf{PH} \subseteq \mathsf{CH} \subseteq \mathsf{PSPACE}$, but we do not know whether $\mathsf{PH} = \mathsf{PSPACE}$.

Each level of the counting hierarchy has a complete problem defined as follows. For every formula $\phi(X)$ with the list X of l free propositional variables, we write

$$\mathsf{C}^m X \phi(X) \longleftrightarrow \sum_{X \in \{0,1\}^l} \phi(X) \geq m \ , \tag{6}$$

where $\phi(X)$ is identified with the function $\phi\colon \{0,1\}^l \to \{0,1\}$ such that $\phi(X) = 1$ when $\phi(X)$ is true. This "counting quantifier" C^m generalizes the usual quantifiers $\exists$ and $\forall$, because $\mathsf{C}^1 = \exists$ and $\mathsf{C}^{2^l} = \forall$. For lists $X_1, \dots, X_n$ of variables and a formula $\phi(X_1, \dots, X_n)$ with all free variables listed, we define

$$\langle \phi(X_1, \dots, X_n), m_1, \dots, m_n \rangle \in \mathsf{C}_n B_{\mathrm{be}} \longleftrightarrow \mathsf{C}^{m_1} X_1 \cdots \mathsf{C}^{m_n} X_n \phi(X_1, \dots, X_n) \ . \tag{7}$$

Lemma 8 ([13, Theorem 7]). *For every* $n \geq 1$*, the problem* $\mathsf{C}_n B_{\mathrm{be}}$ *is* $\mathsf{C}_n\mathsf{P}$*-complete.*

We define the problem $\mathsf{C}_{\log} B_{\mathrm{be}}$ by

$$\langle 0^{2^n}, u \rangle \in \mathsf{C}_{\log} B_{\mathrm{be}} \longleftrightarrow u \in \mathsf{C}_n B_{\mathrm{be}} \ . \tag{8}$$

We show that $\mathsf{C}_{\log} B_{\mathrm{be}}$ is CH-hard and recognized by a logarithmic-height uniform function family, as required in Lemma 7.

Proof (Lemma 7). First we prove that $\mathsf{C}_{\log}B_{\text{be}}$ is CH-hard. For each problem A in CH, there is a constant n such that $A \in \mathsf{C}_n\mathsf{P}$. From Lemma 8, for each $u \in \{0,1\}^*$ there is a polynomial-time function f_n such that $u \in A \leftrightarrow f_n(u) \in \mathsf{C}_n B_{\text{be}}$. So

$$u \in A \longleftrightarrow \langle 0^{2^n}, f_n(u)\rangle \in \mathsf{C}_{\log}B_{\text{be}} \ . \tag{9}$$

Since $\langle 0^{2^n}, f_n(\cdot)\rangle$ is polynomial time computable, A is reducible to $\mathsf{C}_{\log}B_{\text{be}}$.

Next we sketch (the details will be given in the full version of this paper) how to construct, for each formula with n counting quantifiers, a difference equation of height $n+1$ such that the output of its computation (i.e., the number in the bottom right cell) is equal to the value of the formula, and other cells in the last column contain 0. Once we have such a difference equation, we can construct a logarithmic-height uniform function family recognizing $\mathsf{C}_{\log}B_{\text{be}}$, since n is logarithmic in the length of the input of $\mathsf{C}_{\log}B_{\text{be}}$ because of the padding 0^{2^n} in (8). We build up the difference equation recursively. For $n = 0$, it is easy because the value of a formula without quantifiers is polynomial-time computable. Consider the formula $\mathsf{C}^m X\psi(X)$, where $\psi(X)$ has i counting quantifiers, and assume that for each $Y \in \{0,1\}^l$, there exists a difference equation of height $i+1$ that computes $\psi(Y)$. By connecting these equations along the T-axis (i.e., arranging tables like Fig. 3 horizontally), we construct a new difference equation that computes $\sum_Y \psi(Y)$ in the $(i+1)$st row and then uses this sum to compute $\mathsf{C}^m X\psi(X)$ in the next row. To bring the value in the $(i+1)$st row back to 0, we further connect the difference equation computing $-\sum_Y \psi(Y)$, which can be constructed similarly by changing the signs. □

3.3 Families of Real Functions Simulating Difference Equations

We show that certain families of smooth differential equations can simulate PSPACE- or CH-hard difference equations stated in previous section.

Before stating Lemmas 9 and 10, we extend the definition of polynomial-time computability of real function to families of real functions. A machine M *computes* a family $(f_u)_u$ of functions $f_u\colon A \to \mathbf{R}$ indexed by strings u if for any $x \in A$ and any name ϕ_x of x, the function taking v to $M^{\phi_x}(u,v)$ is a name of $f_u(x)$. We say a family of real functions $(f_u)_u$ is polynomial-time if there is a polynomial-time machine computing $(f_u)_u$.

Lemma 9. *There exist a* CH*-hard language L and a polynomial μ, such that for any $k \geq 1$ and polynomials γ, there are a polynomial ρ and families $(g_u)_u$, $(h_u)_u$ of real functions such that $(g_u)_u$ is polynomial-time computable and for any string u:*

(i) $g_u\colon [0,1]\times[-1,1] \to \mathbf{R}$, $h_u\colon [0,1] \to [-1,1]$;
(ii) $h_u(0) = 0$ *and* $Dh_u(t) = g_u(t, h_u(t))$ *for all* $t \in [0,1]$;
(iii) g_u *is of class* $\mathrm{C}^{(\infty,k)}$;
(iv) $D^{(i,0)}g_u(0,y) = D^{(i,0)}g_u(1,y) = 0$ *for all* $i \in \mathbf{N}$ *and* $y \in [-1,1]$;
(v) $\left|D^{(i,j)}g_u(t,y)\right| \leq 2^{\mu(i,|u|)-\gamma(|u|)}$ *for all* $i \in \mathbf{N}$ *and* $j \in \{0,\dots,k\}$;
(vi) $h_u(1) = 2^{-\rho(|u|)}L(u)$.

Lemma 10. *There exist a* PSPACE*-hard language L and a polynomial μ, such that for any polynomial γ, there are a polynomial ρ and families $(g_u)_u$, $(h_u)_u$ of real functions such that $(g_u)_u$ is polynomial-time computable and for any string u satisfying (i)–(vi) of Lemma 9 with $k = 1$.*

In Lemmas 9 and 10 we have the new conditions (iii)–(v) about the derivatives of g_u that were not present in [3, Lemma 4.1]. These conditions will permit the family of functions to be put together in one smooth function in Theorems 1 and 2.

We will prove Lemma 9 using Lemma 7 as follows. Let a function family $(G_u)_u$ be as in Lemma 7, and let $(H_u)_u$ be the family of the solutions of the difference equations given by $(G_u)_u$. We construct h_u and g_u from H_u and G_u such that $h_u(T/2^{q(|u|)}) = \sum_{i=0}^{p(|u|)} H_u(i,T)/B^{d_u(i)}$ for each $T = 0, \dots, 2^{q(|u|)}$ and $Dh_u(t) = g_u(t, h_u(t))$. The polynomial-time computability of $(g_u)_u$ follows from that of $(G_u)_u$. We can prove Lemma 10 from Lemma 6 in the same way.

3.4 Proof of the Main Theorems

Using the function families $(g_u)_u$ and $(h_u)_u$ obtained from Lemmas 9 or 10, we construct the functions g and h in Theorems 1 and 2 as follows. Divide $[0, 1)$ into infinitely many subintervals $[l_u^-, l_u^+]$, with midpoints c_u. We construct h by putting a scaled copy of h_u onto $[l_u^-, c_u]$ and putting a horizontally reversed scaled copy of h_u onto $[c_u, l_u^+]$ so that $h(l_u^-) = 0$, $h(c_u) = 2^{-\rho'(|u|)}L(u)$ and $h(l_u^+) = 0$ where ρ' is a polynomial. In the same way, g is constructed from $(g_u)_u$ so that g and h satisfy (1). We give the details of the proof of Theorem 2 from Lemma 9, and omit the analogous proof of Theorem 1 from Lemma 10.

Proof (Theorem 2). Let L and μ be as Lemma 9. Define $\lambda(x) = 2x + 2$, $\gamma(x) = \mu(x, x) + x\lambda(x)$ and for each u let $\Lambda_u = 2^{\lambda(|u|)}$, $c_u = 1 - 2^{-|u|} + 2\bar{u} + 1/\Lambda_u$, $l_u^{\mp} = c_u \mp 1/\Lambda_u$, where $\bar{u} \in \{0, \dots, 2^{|u|} - 1\}$ is the number represented by u in binary notation. Let ρ, $(g_u)_u$, $(h_u)_u$ be as in Lemma 9 corresponding to the above γ.

We define

$$g\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}, \frac{y}{\Lambda_u}\right) = \begin{cases} \pm \displaystyle\sum_{l=0}^{k} \frac{D^{(0,l)} g_u(t, 1)}{l!} (y-1)^l & \text{if } 1 < y \ , \\ \pm g_u(t, y) & \text{if } -1 \le y \le 1 \ , \\ \pm \displaystyle\sum_{l=0}^{k} \frac{D^{(0,l)} g_u(t, -1)}{l!} (y+1)^l & \text{if } 1 < y \ , \end{cases} \tag{10}$$

$$h\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}\right) = \frac{h_u(t)}{\Lambda_u} \tag{11}$$

for each string u and $t \in [0, 1)$, $y \in [-1, 1]$. Let $g(1, y) = 0$ and $h(1) = 0$ for any $y \in [-1, 1]$.

It can be shown similarly to the Lipschitz version [3, Theorem 3.2] that g and h satisfy (1) and g is polynomial-time computable. Here we only prove that g is of class $\mathrm{C}^{(\infty,k)}$. We claim that for each $i \in \mathbf{N}$ and $j \in \{0,\dots,k\}$, the derivative $D_1^i D_2^j g$ is given by

$$D_1^i D_2^j g\left(l_u^\mp \pm \frac{t}{\Lambda_u}, \frac{y}{\Lambda_u}\right) = \begin{cases} \pm\Lambda_u^{i+j} \sum_{l=j}^{k} \frac{D^{(i,l)} g_u(t,1)}{(l-j)!}(y-1)^l & \text{if } y < -1 \ , \\ \pm\Lambda_u^{i+j} D^{(i,j)} g_u(t,y) & \text{if } -1 \le y \le 1 \ , \\ \pm\Lambda_u^{i+j} \sum_{l=j}^{k} \frac{D^{(i,l)} g_u(t,-1)}{(l-j)!}(y+1)^l & \text{if } 1 < y \end{cases} \tag{12}$$

for each $l_u^\mp \pm t/\Lambda_u \in [0,1)$ and $y/\Lambda_u \in [-1,1]$, and by $D_1^i D_2^j g(1,y) = 0$. This is verified by induction on $i+j$. The equation (12) follows from calculation (note that this means verifying that (12) follows from the definition of g when $i = j = 0$; from the induction hypothesis about $D_2^{j-1} g$ when $i = 0$ and $j > 0$; and from the induction hypothesis about $D_1^{i-1} D_2^j g$ when $i > 0$). That $D_1^i D_2^j g(1,y) = 0$ is immediate from the induction hypothesis if $i = 0$. If $i > 0$, the derivative $D_1^i D_2^j g(1,y)$ is by definition the limit

$$\lim_{s \to 1-0} \frac{D_1^{i-1} D_2^j g(1,y) - D_1^{i-1} D_2^j g(s,y)}{1-s} \ . \tag{13}$$

This can be shown to exist and equal 0, by observing that the first term in the numerator is 0 and the second term is bounded, when $s \in [l_u^-, l_u^+]$, by

$$\begin{aligned} |D_1^{i-1} D_2^j g(s,y)| &\le \Lambda_u^{i-1+j} \sum_{l=j}^{k} |D^{(i-1,l)} g_u| \cdot (\Lambda_u + 1)^l \\ &\le \Lambda_u^{i-1+j} \cdot k \cdot 2^{\mu(i-1,|u|)-\gamma(|u|)} \cdot (2\Lambda_u)^k \\ &\le 2^{(i-1+j+k)\lambda(|u|)+2k+\mu(i-1,|u|)-\gamma(|u|)} \le 2^{-2|u|} \le 2^{-|u|+1}(1-s) \ , \end{aligned} \tag{14}$$

where the second inequality is from Lemma 9 (v) and the fourth inequality holds for sufficiently large $|u|$ by our choice of γ. The continuity of $D_1^i D_2^j g$ on $[0,1) \times [-1,1]$ follows from (12) and Lemma 9 (iv). The continuity on $\{1\} \times [-1,1]$ is verified by estimating $D_1^i D_2^j g$ similarly to (14). □

4 Complexity of Operators

Both Theorems 1 and 2 state the complexity of the solution h under the assumption that g is polynomial-time computable. But how hard is it to "solve" differential equations, i.e., how complex is the operator that takes g to h? To make this question precise, we need to define the complexity of operators taking real functions to real functions.

Recall that, to discuss complexity of real functions, we used string functions as names of elements in $\mathbf{R}$. Such an encoding is called a *representation* of $\mathbf{R}$. Likewise, we now want to encode real functions by string functions to discuss complexity of real operators. In other words, we need to define representations of the class $\mathrm{C}_{[0,1]}$ of continuous functions $h\colon [0,1] \to \mathbf{R}$ and class $\mathrm{CL}_{[0,1]\times[-1,1]}$

of Lipschitz continuous functions $g\colon [0,1] \times [-1,1] \to \mathbf{R}$. The notions of computability and complexity depend on these representations. Following [11], we use $\delta_\square$ and $\delta_{\square\mathrm{L}}$ as the representations of $\mathrm{C}_{[0,1]}$ and $\mathrm{CL}_{[0,1]\times[-1,1]}$, respectively. It is known that $\delta_\square$ is the canonical representation of $\mathrm{C}_{[0,1]}$ in a certain sense [15], and $\delta_{\square\mathrm{L}}$ is the representation defined by adding to $\delta_\square$ the information on the Lipschitz constant.

Since these representations use string functions whose values have variable lengths, we use *second order polynomials* to bound the amount of resources (time and space) of machines [11], and this leads to the definitions of second-order complexity classes (e.g. **FPSPACE**, polynomial-space computable), reductions (e.g. $\le_{\mathrm{W}}$, polynomial-time Weihrauch reduction), and hardness. Combining them with the representations of real functions mentioned above, we can restate our theorems in the constructive form as follows.

Let *ODE* be the operator mapping a real function $g \in \mathrm{CL}_{[0,1]\times[-1,1]}$ to the solution $h \in \mathrm{C}_{[0,1]}$ of (1). The operator *ODE* is a partial function from $\mathrm{CL}_{[0,1]\times[-1,1]}$ to $\mathrm{C}_{[0,1]}$ (it is partial because the trajectory may fall out of the interval $[-1,1]$, see the paragraph following Theorem 2). In [11, Theorem 4.9], the $(\delta_{\square\mathrm{L}}, \delta_\square)$-**FPSPACE**-$\le_{\mathrm{W}}$-completeness of *ODE* was proved by modifying the proof of the results in the third row of Table 1. Theorem 1 can be rewritten in a similar way. That is, let ODE_k be the operator *ODE* whose input is restricted to class $\mathrm{C}^{(\infty,k)}$. Then we have:

Theorem 11. *The operator ODE_1 is $(\delta_{\square\mathrm{L}}, \delta_\square)$-***FPSPACE***-$\le_{\mathrm{W}}$-complete.*

To show this theorem, we need to verify that the information used to construct functions in the proof of Theorem 1 can be obtained easily from the inputs. We omit the proof since it does not require any new technique. Theorem 11 automatically implies Theorem 1 because of [11, Lemmas 3.7 and 3.8].

In contrast, the polynomial-time computability in the analytic case (the last row of Table 1) is *not* a consequence of a statement based on $\delta_\square$. It is based on the calculation of the Taylor coefficients, and hence we only know how to convert the Taylor sequence of g at a point to that of h. In other words, the operator *ODE* restricted to the analytic functions is not $(\delta_{\square\mathrm{L}}, \delta_\square)$-**FP**-computable, but $(\delta_{\mathrm{Taylor}}, \delta_{\mathrm{Taylor}})$-**FP**-computable, where δ_{Taylor} is the representation that encodes an analytic function using its Taylor coefficients at a point in a suitable way. More discussion on representations of analytic functions and the complexity of operators on them can be found in [16].

Acknowledgments. The writing of this paper was helped greatly by the comments and discussions at the 10th EATCS/LA Workshop on Theoretical Computer Science, held in Kyoto in early 2012, where the authors reported preliminary results. This work was supported in part by *Kakenhi* (Grant-in-Aid for Scientific Research, Japan) 23700009 and by the *Marie Curie International Research Staff Exchange Scheme Fellowship* 294962 within the 7th European Community Framework Programme.

References

1. Ko, K.: Complexity Theory of Real Functions. Birkhäuser Boston (1991)
2. Weihrauch, K.: Computable Analysis: An Introduction. Springer (2000)
3. Kawamura, A.: Lipschitz continuous ordinary differential equations are polynomial-space complete. Computational Complexity 19(2), 305–332 (2010)
4. Pour-El, M., Richards, I.: A computable ordinary differential equation which possesses no computable solution. Annals of Mathematical Logic 17(1-2), 61–90 (1979)
5. Coddington, E., Levinson, N.: Theory of Ordinary Differential Equations. McGraw-Hill (1955)
6. Ko, K.: On the computational complexity of ordinary differential equations. Information and Control 58(1-3), 157–194 (1983)
7. Miller, W.: Recursive function theory and numerical analysis. Journal of Computer and System Sciences 4(5), 465–472 (1970)
8. Müller, N.: Uniform Computational Complexity of Taylor Series. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 435–444. Springer, Heidelberg (1987)
9. Ko, K., Friedman, H.: Computing power series in polynomial time. Advances in Applied Mathematics 9(1), 40–50 (1988)
10. Kawamura, A.: Complexity of initial value problems. Fields Institute Communications (to appear)
11. Kawamura, A., Cook, S.: Complexity theory for operators in analysis. ACM Transactions on Computation Theory 4(2), Article 5 (2012)
12. Bournez, O., Graça, D.S., Pouly, A.: Solving Analytic Differential Equations in Polynomial Time over Unbounded Domains. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 170–181. Springer, Heidelberg (2011)
13. Wagner, K.: The complexity of combinatorial problems with succinct input representation. Acta Informatica 23(3), 325–356 (1986)
14. Torán, J.: Complexity classes defined by counting quantifiers. Journal of the ACM 38(3), 752–773 (1991)
15. Kawamura, A.: On function spaces and polynomial-time computability. Dagstuhl Seminar 11411: Computing with Infinite Data (2011)
16. Kawamura, A., Müller, N., Rösnick, C., Ziegler, M.: Uniform polytime computable operators on univariate real analytic functions. In: Proceedings of the Ninth International Conference on Computability and Complexity in Analysis (2012)

The Lower Reaches of Circuit Uniformity

Christoph Behle[1], Andreas Krebs[1], Klaus-Jörn Lange[1], and Pierre McKenzie[2]

[1] Wilhelm-Schickard-Institut, Universität Tübingen
{behlec,krebs,lange}@informatik.uni-tuebingen.de
[2] DIRO, Université de Montréal*
mckenzie@iro.umontreal.ca

Abstract. The effect of severely tightening the uniformity of Boolean circuit families is investigated. The impact on NC^1 and its subclasses is shown to depend on the characterization chosen for the class, while classes such as P appear to be more robust. Tightly uniform subclasses of NC^1 whose separation may be within reach of current techniques emerge.

1 Introduction

Motivation. Uniformity is imposed on Boolean circuit families in order for circuit families to define classes of languages that correspond to machine-based classes. For example, logspace-uniform and polytime-uniform Boolean circuit families of polynomial size [Bor77] capture the class P, while non-uniform circuit families of constant size recognize undecidable languages.

Uniformity notions more permissive than the circuit resources under study were considered in the literature (e.g. [All89]). But tighter and tighter notions were needed to capture low complexity classes. Borodin [Bor77] and Cook [Coo79] first showed the usefulness of enforcing uniformity by means of space-bounded Turing machines computing circuit descriptions. This worked well for NC^2 and above. Inspired by Goldschlager [Gol78], Ruzzo [Ruz81] then tied uniformity to circuit connectivity queries. Ruzzo defined an ALOGTIME notion of uniformity under which NC^1 meaningfully equals ALOGTIME. Yet tighter notions were needed to investigate $\mathrm{AC}^0 \subset \mathrm{ACC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1$. Barrington, Immerman and Straubing [BIS90] thus developed DLOGTIME-uniformity. They proved that the model-theoretic notion introduced by Immerman [Imm87] and the Turing machine-based notion studied by Buss [Bus87] were equivalent to their own.

Roy and Straubing [RS07] later triggered the need for an even stronger notion of uniformity than DLOGTIME. This requires some explaining, because DLOGTIME is surely the lowest meaningful Turing machine-based complexity class imaginable.

Motivation Continued: Enters Descriptive Complexity. The language of words $w \in \{a,b\}^\star$ having no b at an even position can be described by the intuitive formula $\neg\exists i\big(\mathsf{Even}(i) \wedge Q_b(i)\big)$. In such a formula, the variables range over positions in w, the predicate Q_σ for $\sigma \in \{a,b\}$ holds at i iff $w_i = \sigma$, and the *numerical* predicate Even holds at i iff i is even. This example is a first-order

[*] Supported by the Natural Sciences and Engineering Research Council of Canada.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 590–602, 2012.

formula, more precisely a $\mathrm{FO}[<, \mathsf{Even}]$ formula, where the numerical predicates allowed are listed and they include "$i < j$" for the formalism to be able to describe words, i.e., (ordered) sequences of letters.

Descriptive complexity based on ExtFO (our appellation here for FO extended with generalized quantifiers) is tied to circuit complexity in two important ways. First, low circuit complexity classes have crisp ExtFO characterizations. Second, FO provides the desired notions of uniformity finer than DLOGTIME.

Indeed a language is in non-uniform AC^0 iff it can be described by a $\mathrm{FO}[\mathsf{arb}]$ formula [Imm87], where arb means that "any arbitrary set of numerical predicates" is allowed. A language is in non-uniform TC^0 iff it can be described by a $\mathrm{MAJ}[\mathsf{arb}]$ formula [BIS90], where MAJ refers to replacing "$\exists$" and "$\forall$" with the "there exist more than half of the possible positions i at which the formula holds" quantifier. And a language is in non-uniform NC^1 iff it can be described in $S_5 + \mathrm{FO}[\mathsf{arb}]$, where $S_5 + \mathrm{FO}$ refers to allowing, in addition to "$\exists$" and "$\forall$", a (Lindström) quantifier testing whether a sequence of five-point permutations associated with each position i composes to the identity permutation [Bar89].

Strikingly, replacing $[\mathsf{arb}]$ above with $[+, \mathsf{x}]$, or equivalently $[<, \mathsf{bit}]$ as shown in [BIS90], imposes DLOGTIME-uniformity. For example, $\mathrm{FO}[+, \mathsf{x}]$ precisely equals DLOGTIME-uniform AC^0 and $S_5 + \mathrm{FO}[+, \mathsf{x}]$ equals DLOGTIME-uniform NC^1. The numerical predicates available to the ExtFO description of a circuit-based class of languages thus regulate the uniformity of the circuit families.

Roy and Straubing proved [RS07] that any regular language expressed in $\mathrm{MOD}_q + \mathrm{FO}[<, +]$ can be re-expressed without the non-regular[1] numerical predicate "+", where MOD_q refers to allowing the "there exist m modulo q positions i at which the formula holds" quantifiers. Roy and Straubing asked whether $\mathrm{MOD}_q + \mathrm{FO}[<, +]$ also has a circuit characterization. This was answered by the first and third authors who proved, using a new encoding for circuit connections, that $\mathrm{ExtFO}[<, \mathsf{X}]$ is meaningfully captured by $\mathrm{FO}[<, \mathsf{X}]$-uniform circuits, for any reasonable set X of numerical predicates [BL06].

Motivation Concluded. Meaningful uniformity notions even tighter than DLOGTIME uniformity thus abound: examples are $\mathrm{FO}[<, +]$-uniformity and $\mathrm{FO}[<]$-uniformity. In the $\mathrm{FO}[<, +]$-uniform world, Roy and Straubing thus separated classes that are only *conjectured* to differ in the more usual $\mathrm{FO}[+, \mathsf{x}]$-uniform, i.e., DLOGTIME-uniform, world.

Our main motivation is to examine the impact of imposing $\mathrm{FO}[<]$-uniformity. Do the separations conjectured in the DLOGTIME-uniform world hold here? What happens to TC^0 and to the following well-known characterizations of NC^1,

Characterization	Depth	Fan-in	Size	Gates used
NC^1	$O(\log n)$	constant	poly	AND, OR, NOT
$\mathrm{AC}^0(M)$	$O(1)$	poly	poly	AND, OR, M
$\mathrm{AC}^0(\mathbb{D}_+)$	$O(1)$	poly	poly	AND, OR, D_+
$\mathrm{AC}^0(\mathbb{D}_+)_{\mathrm{LIN}}$	$O(1)$	linear	poly	AND, OR, D_+

[1] A numerical predicate is said to be regular if a finite automaton can compute it; much of the structure of NC^1 hinges on a "*regularity*" conjecture [Str94, IX.3.4].

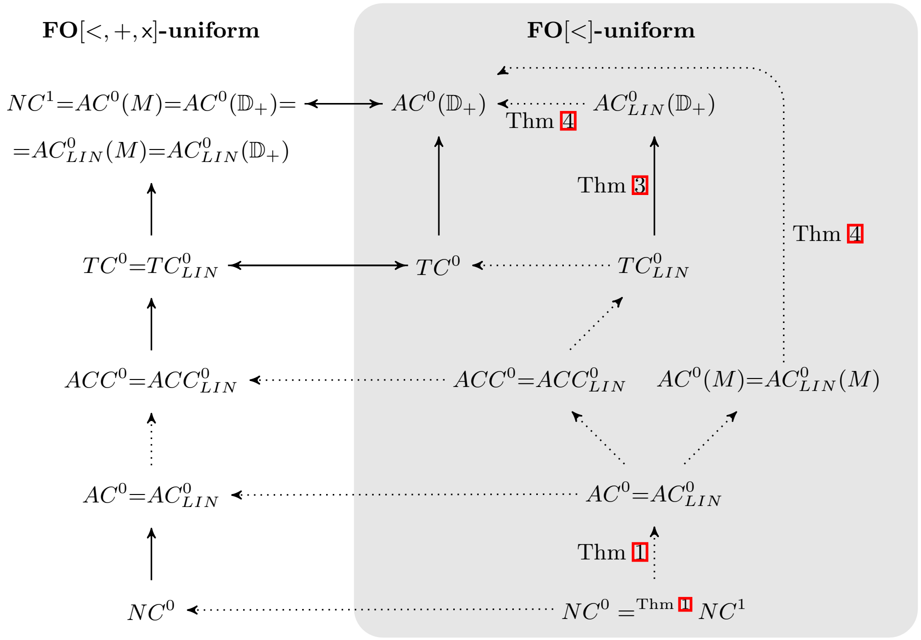


Fig. 1. Except in the case of NC^1 = ALOGTIME, FO[ ]-uniformity here refers to direct connection language expressivity under unary shuffled encoding (see Section 2). A solid arc, a bidirectional arc and a dashed arc from A to B denote $A \subseteq B$, $A = B$ and $A \subset B$ respectively. Arrows with no labels were known prior to this paper.

where $\mathrm{AC}^0(M)$ is the AC^0-closure of a set of word problems over finite monoids from a set M that contains at least one non-solvable monoid, e.g. A_5 [Bar89], where $\mathbb{D}_+$ is a variant of the Boolean formula value problem [Bus87, BCGR92] (see Section 2) and where $\mathrm{AC}^0(\mathbb{D}_+)_{\mathrm{LIN}}$ is $\mathrm{AC}^0(\mathbb{D}_+)$ with linear fanin gates?

Our Results. See Figure 1. Together with some further observations, we deduce the following properties of the FO[<]-uniform world:

- the logarithmic depth characterization of NC^1 collapses down to NC^0; more generally (Theorem 1), any class described by sublinear depth circuits, such as AC^1 or NC, collapses down to its constant-depth level;
- by contrast, adding to "$i < j$" the numerical predicate "$i = 2j$", hence a predicate weaker than + or x, restores ALOGTIME (Theorem 6);
- $\mathrm{AC}^0(A_5) \subset \mathrm{ACC}^0(A_5) \subset$ REGULAR [BIS90, BL06], thus $\mathrm{AC}^0(A_5) \not\supseteq \mathrm{TC}^0$;
- ALOGTIME = $\mathrm{AC}^0(\mathbb{D}_+)$ [Bus87], thus $\mathrm{AC}^0(\mathbb{D}_+) \supseteq \mathrm{TC}^0$;
- $\mathrm{AC}^0(\mathbb{D}_+)_{LIN} \supseteq \mathrm{TC}^0{}_{LIN}$ (Theorem 3);
- $\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$ can neither express the x nor the bit numerical predicate; since the presence of the bit predicate is a major hurdle in lower bound proofs, this suggests attempting to separate $\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$ from $\mathrm{TC}^0{}_{LIN}$ as the next target towards understanding the relationship between TC^0 and NC^1;
- polynomial size Boolean circuits capture P (Theorem 5).

FO[<]-uniformity is defined here as in [BL06] using *unary* shuffled encoding. For purposes of comparison, we also define a FO[+1]-uniform$_{\text{bin}}$ world, where "<" is replaced with the weaker "+1" and circuit parameters are expressed in binary notation (following [BCGR92]). In that world, NC^1 does contain ALOGTIME (Theorem 6) and polynomial size circuits still capture P.

2 Preliminaries and Definitions

We assume familiarity with circuits and only recall some fundamental definitions. For $i \geq 0$, the class NC^i is the set of languages recognized by circuit families of depth $O((\log n)^i)$ and polynomial size built from bounded fan-in AND, OR and NOT gates. The class AC^i is obtained instead when the AND and OR gates have unbounded fan-in and ACC^i is further obtained when unbounded fan-in MOD_q gates are also permitted. The class TC^i is the set of languages recognized by circuit families of depth $O((\log n)^i)$ and polynomial size built from unbounded fan-in MAJORITY gates. We write $\mathrm{AC}^0(G)$ for the class AC^0 in which the circuits are additionally equipped with unbounded fan-in gates of type G.

Definition 1 (Direct connection language of a circuit family). *Let C_n be a circuit with n inputs and size $\leq n^c$. We label the gates by c-tuples of numbers from 1 to n. Then the direct connection language of C_n is*

$$\begin{aligned} L_{C_n} = \{\langle t, \boldsymbol{a}, \boldsymbol{b}, n\rangle \mid & \textit{the gate } g \textit{ labeled by } \boldsymbol{a} \textit{ has type } t \textit{ and, either,} \\ & t = \mathsf{input} \textit{ and } g \textit{ queries the input bit } \boldsymbol{b}_1 \in \{1, \ldots, n\}, \textit{ or} \\ & t \in \{\textit{true}, \textit{false}\} \textit{ and } g \textit{ is assigned the value } t, \textit{ or} \\ & t \in \{\neg, \wedge, \vee, G\} \textit{ and the gate labeled } \boldsymbol{b} \textit{ is a predecessor of } g\}. \end{aligned}$$

The direct connection language of a sequence of circuits $C = (C_1, \ldots)$ is $L_C = \bigcup_n L_{C_n}$. The predecessors of a gate are fed into the gate in ascending order of their numbers (w.l.g., since a gate g input to a gate g' can be assigned any number smaller than the numbering of g' by the insertion of a dummy gate along the edge (g, g')). The output gate is always labeled by $(1, \ldots, 1)$.

The language L_{C_n} may describe gates having no path to the output, or unreachable from the inputs, or both. Such gates do contribute to the size of C_n. The depth of C_n is defined as the longest path in the graph of the circuit, regardless of whether this path connects an input gate to the output gate. We will use numbers to encode the type of a gate in t.

We have yet to fix an encoding that will turn the above direct connection language into a set of words over a fixed alphabet. The *unary n-encoding* of a number $1 \leq i \leq n$ is defined as the word $a^i b^{n-i}$ over $\Sigma = \{a, b\}$.

Given a sequence of words $w_1, \ldots, w_c$ of a common length n over a common alphabet Σ, the *shuffle* of this sequence is the unique word u of length n over Σ^c, defined by setting the i-th letter of u to $(\sigma_1, \ldots, \sigma_c)$ iff σ_j is the i-th letter of w_j for $1 \leq j \leq c$.

Definition 2 (Unary shuffled encoding [BL06]). *The* unary shuffled n-encoding *of a sequence $\alpha_1, \ldots, \alpha_k$ of numbers between* 1 *and n is the shuffle of the sequence of unary n-encodings of $\alpha_1, \ldots, \alpha_k$. When n is understood, this shuffle is denoted $\langle \alpha_1, \ldots, \alpha_k \rangle$ (and is a word of length n over $\{a, b\}^k$).*

Definition 3 (FO[X]-uniform circuit family). *For* X *a set of numerical predicates, i.e., a set of relations of various arities over $\mathbb{N}$, we say that a circuit family $(C_n)_{n \geq 1}$ is* FO[X]-uniform *if the following language is expressible in* FO[X]*:*

$$\bigcup_{n \geq 1} \{w \in (\{a, b\}^{2c+2})^* \colon\ w = \textit{unary shuffled } n\textit{-encoding of some } \langle t, \boldsymbol{a}, \boldsymbol{b}, n \rangle \in L_{C_n}\}.$$

Remark 1. By [BL06], FO[$<, +, \times$]-uniform $\mathrm{AC}^0(G)$, ACC^0 and TC^0 respectively equal DLOGTIME-uniform $\mathrm{AC}^0(G)$, ACC^0 and TC^0 as defined in [BIS90]. In particular, FO[$<, +, \times$]-uniform $\mathrm{AC}^0(A_5)$ equals what is commonly referred to as DLOGTIME-uniform NC^1 and thus ALOGTIME [BIS90]. To clear possible confusion, recall that to obtain DLOGTIME-uniform NC^1 from a DLOGTIME criterion applied to logarithmic depth circuits, the *extended* connection language is used [Ruz81]. It is known that U_D-$\mathrm{NC}^1 \supseteq$ ALOGTIME, where U_D-NC^1 is defined by DLOGTIME recognition of L_C for logarithmic depth circuit families C, but equality is still open (see [Vol99, P. 162]).

When dealing with the Dyck language $\mathbb{D}_1$ over one pair of parentheses we will use the letters $\{a, b\}$ instead of $\{(,)\}$ to improve readability.

We define now a formal language version of the formula value problem. We encode complete binary trees, i.e.: trees where each inner node has exactly two predecessors, by a traversal from left to right. A left edge is labeled by a and a right edge is labeled by b. This gives a one-to-one correspondence between complete binary trees and the Dyck language $\mathbb{D}_1 \subset \{a, b\}^*$.

We decided in favor of labelling the edges and against the more usual labelling of the vertices, which would lead to the well known representation of complete binary trees by the Lukasiewicz language.

A tree labelled by Boolean functions and constants evaluates either to *true* or to *false*. In this way the Dyck set $\mathbb{D}$ is divided into the two disjoint subsets $\mathbb{D} = \mathbb{D}_+ \cup \mathbb{D}_-$ where $\mathbb{D}_+$ consists in those elements of $\mathbb{D}$ which represent a tree (labelled with the NAND-function and the constant *true*) which evaluates to *true* and $\mathbb{D}_-$ contains those which evaluate to *false*. Thus $\mathbb{D}_+$ is a special formulation of the Boolean formula value problem which makes $\mathbb{D}_+$ NC^1-complete.

Definition 4 ($\mathbb{D}_+$ language). *Any word in the Dyck language $\mathbb{D}_1$ corresponds to a tree as defined above. If we let every leaf of the tree be a true node and every inner node be a* NAND*, the tree is a formula evaluating either to true or false. We let $\mathbb{D}_+ \subset \{a, b\}^*$ be the set of words that are in the Dyck language and whose corresponding formula evaluates to true.*

We now define two gate types based on languages that are complete for NC^1, even in the case of DLOGTIME-uniformity.

Definition 5 **(A_5 Gate).** *Let A_5 be the alternating group over 5 elements and $g_0, g_1 \in A_5$ be two 5 cycles that span the whole group. An A_5 gate with k Boolean inputs $x_1, \dots, x_k$ evaluates to 1 if $g_{x_1} \dots g_{x_k} = 1$, otherwise to 0.*

Recall that we write the Dyck language over the alphabet $\{a, b\}$.

Definition 6 **($\mathbb{D}_+$ Gate).** *A $\mathbb{D}_+$ gate with k Boolean inputs $x_1, \dots, x_k$ evaluates to 1 if replacing every 0 by a and every 1 by b in the word $x_1 \cdots x_k$ yields a word in $\mathbb{D}_+$.*

We assume some familiarity with first order descriptive complexity. We follow Straubing and use his semantics based on $\mathcal{V}$-structures [Str94, P. 14]. We write FO to denote first order logic and write the set of allowed numerical predicates in brackets. We will use the binary predicates bit, $<$, $+1$, and x2. Since the value of a numerical predicate depends only on the position of the variables and the word length, we will freely switch between variables and natural numbers denoting their positions. To make this transition clearer we write "$x = i$" for a variable x and a natural number i when x points to the i-th position. The predicate $\mathsf{bit}(i, j)$ is true if the i-th bit of the binary representation of j is a 1. The successor predicate $+1(i, j)$ is true $i = j + 1$. The double predicate $\mathsf{x2}(i, j)$ is true if $i = 2j$. Recall that FO[<] only describes regular languages (it in fact captures the aperiodic regular languages [MP71]). The class FO[+1] is a proper subclass of FO[<] that in fact captures the threshold testable languages [Tho78].

The class FO can also be extended by adding additional quantifiers. We write FO + MAJ, FO + A_5, and FO + M, resp., for the class FO which is also equipped with the majority quantifier, with an A_5-quantifier, and with arbitrary finite monoid quantifiers, respcectively.

3 FO[<]-Uniform Logarithmic Depth Circuits

Our first theorem shows that FO[<]-uniform NC^1 is restrictive as the class collapses down to NC^0. If we add a simple numerical predicate like x2 we obtain full DLOGTIME-uniform NC^1.

To prove our first theorem we show that FO[<] cannot express an edge relation such that the resulting graph has paths of length $\Theta(\log n)$. We first explain why we need only to consider the expressiveness of FO[<] in terms of numerical predicates.

The expressive power of FO[<] is well understood. One important restriction of FO[<] is that a fixed formula can only count up to a constant. Beyond this constant it can only check the relative ordering. It is for example known that for quantifier depth d a formula cannot distinguish between the words a^{2^d} and a^{2^d+1}.

We use the following observation, see for example [Str94][p. 79].

Lemma 1. *Let ϕ be a formula of quantifier depth d over the alphabet Σ, let $u, w \in \Sigma^*$, and let $v \in \Sigma$. Then $uv^{2^d-1}w \models \phi$ iff $uv^{2^d}w \models \phi$.*

This allows us to prove the following theorem:

Theorem 1. *Let X be any circuit class that limits the depth $d(n)$ to be sublinear, i.e. $d(n) \in o(n)$, and X' be the same circuit class with the restriction that the depth is constant.* FO[<]-uniform X=FO[<]-uniform X'.

Proof. Let $(C_n)_n$ be a family of circuits in FO[<]-uniform X of depth $l(n)$, the labels of $(C_n)_n$ are k-tuples of numbers, and the connection language is recognized by a FO[<] formula ϕ of quantifier depth d.

We prove the theorem by showing that one cannot define a circuit of depth sublinear but not constant, i.e. $l(n)$ is sublinear. There is a value N_0 such that for all $n > N_0$ we have $n \geq (l(n) \cdot k + 1) \cdot (2^d + 1)$. By induction we show for all $n \geq N_0$ that $l(n) \leq l(N_0)$. For the basis $l(N_0) \leq l(N_0)$ there is nothing to prove.

Choose any $n > N_0$. Let $(G_1, \ldots, G_{l(n)})$ be a sequence of gates in C_n, such that for $1 < i \leq l(n)$ the gate G_i is an input gate of G_{i-1}. So the gates form a "path" in the circuit C_n. We let L_i be the gate label of G_i, which is a shuffled unary encoding of k numbers. Let w be the shuffled encoding of all L_i for $i = 1, \ldots, l(n)$, and let Σ be the corresponding alphabet. So w is a word which is the unary shuffled encoding of $l(n) \cdot k$ numbers. We have a formula ψ for our family of circuits that checks if a word over Σ^* is the shuffled encoding of a path of length $l(n)$ in any one of the circuits $C_{n'}$, by using a $(l(n) - 1)$-ary conjunction of the connection formula ϕ. The depth of ψ is d.

Let $\alpha_1, \ldots, \alpha_{l(n)\cdot k}$ be the ordered set of the $l(n) \cdot k$ numbers. The shuffled encoding of these numbers will have the same letter in each of these intervals, i.e. for $i < i'$ if there does not exist a j such that $i \leq \alpha_j \leq i'$, then the i-th letter equals i'-th letter, i.e. $w_i = w_{i'}$. Since $n \geq (l(n) \cdot k + 1) \cdot (2^d + 1)$ there exists a factor word in w of the form σ^{2^d} for $\sigma \in \Sigma$, i.e. $w = u\sigma^{2^d}v$ for $u, v \in \Sigma^*$.

So we will apply Lemma 1 to w and ψ, and obtain a word w' of length $n - 1$. But this word can be interpreted as the shuffled encoding of the labels of a gates sequence $(G'_1, \ldots, G'_{l(n)})$ in C'_{n-1}. Also by Lemma 1 we know that no formula of depth d can distinguish w and w', hence G'_i is an input gate of G'_{i-1} in C_{n-1}. It follows that $l(n) \leq l(n - 1)$ and by the induction hypothesis $l(n) \leq l(N_0)$. □

In the previous theorem if we choose G_1 to be the output gate, we would have that G'_1 is also the output gate, hence not just some irrelavent gates that do not influence the output gate generate the problem, but an actually path form the output gate to an input gate can only have constant length.

Corollary 1. *In the world of* FO[<]*-uniformity,* $\mathrm{NC}^i = \mathrm{NC}^0$*,* $\mathrm{AC}^i = \mathrm{AC}^0$ *and* $\mathrm{TC}^i = \mathrm{TC}^0$ *hold for every* $i \geq 1$*.*

In view of the inability for a FO[<]-uniform class to recognize polylogarithmic depth properly, one might consider adding a unary predicate such as $\log_n(x)$ defined to be true if $\log(n) = x$.

Using the same idea as above one can show that the output gate labeled by $(l_1, \ldots, l_k)$ can only access input positions in a polylogarithmic range around $(l_1, \ldots, l_k)$, since we have a path from the gate labeled $(l_1, \ldots, l_k)$ to the input gate at position p_i of polylogarithmic length. Hence the same proof shows that one cannot obtain FO[$<$, log]-uniform NC^i circuits that are able to recognize the language 1^*.

While FO[$<$] cannot describe circuits of logarithmic length, adding a simple binary predicate like $\mathsf{x2}$, which is much weaker than for example $+$, already allows to describe DLOGTIME uniform circuits (see Theorem 6).

We obtain a dichotomy for our uniformity definitions and circuits of logarithmic depth. The classes result either in subclasses of NC^0 or contain DLOGTIME-uniform NC^i. The fact that even low uniform circuit classes capture DLOGTIME-uniform NC^1 stems from its equivalence to ALOGTIME. Turing machines are uniform and operate only locally. As noted in [BL06] this also allows tightly uniform circuit characterizations for polynomial time. In Section 5 we discuss how this can be extended to a general framework.

4 FO[$<$]-Uniform Constant-Depth NC^1 Characterizations

In this section we will not consider log depth circuits but constant depth circuits equipped with gates that compute NC^1 complete languages. We consider $\mathbb{D}_+$ and the word problem over A_5 as complete problems.

Extending AC^0 by A_5 gates gives a proper subclass of the regular languages.

Theorem 2. *Let M be any subset of monoids, then* FO[$<$]-uniform $\mathrm{AC}^0(M) \subseteq$ REGULAR*, where equality only occurs if every finite monoid can be simulated by (i.e., is a homomorphic image of a submonoid of) a monoid from M.*

Proof. This follows by translating the circuit into a FO + M[$<$] formula and applying Theorem 11.6 from [BIS90]. Here FO + M stands for first order logic equipped with monoid quantifiers as defined in [BIS90]. □

The same argument does not only show that FO[$<$]-uniform $\mathrm{AC}^0(M)$ circuits are contained in the regular languages but that even FO[$<$]-uniform $\mathrm{ACC}^0(M)$ recognize only a subset of the regular languages. The only way to obtain something containing an acceptably large subclass of TC^0 seems to be the bit predicate, but this immediately yields uniform NC^1.

So instead of choosing gates based on finite non-solvable groups, we choose a gate type that corresponds to the Boolean formula value problem. It is known [BL06] that that FO[$<$]-uniform TC^0 is separated from FO[$<$]-uniform TC^0 with linear fan-in. While the former can simulate the bit predicate and is hence equal to DLOGTIME-uniform TC^0 the latter equals FO+MAJ[$<$] and cannot simulate the bit predicate.

Yet we can show that inclusion under DLOGTIME uniformity remains valid under FO[$<$] uniformity:

Theorem 3. FO[$<$]-uniform $\mathrm{AC}^0(\mathbb{D}_+)_{LIN} \supseteq$ FO[$<$]-uniform TC^0_{LIN}.

We mention that MOD_q gates can be simulated in FO[$<$]-uniform TC^0_{LIN}. Together with the previous theorem it follows that FO[$<$]-uniform $\mathrm{AC}^0(\mathbb{D}_+) \supseteq$

FO[<]-uniform $\mathrm{TC}^0 \supseteq$ FO[<]-uniform ACC^0. These inclusions remain valid for the corresponding circuit classes with linear fan-in. The following theorem shows that $\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$ is strictly weaker than NC^1, but it is still not clear whether it is contained in (even non uniform) TC^0.

The following theorem is a consequence of Theorem 4.16 in [LMSV01], where it is shown that only semilinear predicates can be computed by groupoidal qunatifiers using only the order predicate.

Theorem 4. *The predicate* x *(and hence* bit*) cannot be expressed in* FO[<]-uniform $\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$.

Theorem 3 and Theorem 4 highlight the important difference between $\mathrm{AC}^0(\mathbb{D}_+)$ and $\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$; indeed the former is able to simulate bit and thus equals NC^1. Note that superlinear fan-in of gates corresponds to quantifiers over tuples of variables in the logic world.[2]

Summarizing we are able to say: in contrast to A_5 gates, using $\mathbb{D}_+$ gates yields a class that is weaker than DLOGTIME uniform NC^1 but contains uniform subclasses of TC^0_{LIN}. Hence, FO[<]-uniform $\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$ is a candidate subclass of NC^1 worthy of attempts to separate it from TC^0.

5 A Guide to Minimal Uniformity

In Section 3 we noticed that log depth circuits with very tight uniformity can already simulate circuits which are defined by a more powerful uniformity. Ruzzo already showed that for larger classes in NC different uniformity notions coincide. This phenomenon is much more general and we explore it in this section. We start by showing that polynomial time admits very uniform circuits over the standard Boolean gates. Recall that a polynomial size circuit family is P-uniform if L_{C_n} can be listed by a Turing machine in time polynomial in n.

Theorem 5. P*-uniform and* FO[+1]-uniform *polynomial size circuits each capture the class* P*.*

An obvious extension to Theorem 5 is to consider more general complexity classes of Turing machines. Consider a complexity class $\mathcal{M}$ defined by time and space bounds. When simulating a TM in $\mathcal{M}$ as a circuit then time translates to depth and space to width. If one equips FO[<] with unary predicates that allow to check the depth and width bounds, it is possible to perform the construction without much overhead. Therefore, a (deterministic) Turing machine can be simulated by very uniform circuits. If now conversely such a circuit can be evaluated by a TM in $\mathcal{M}$ then any $\mathcal{M}$-uniform $\mathcal{C}$ circuit can be converted to a FO[<, $\mathsf{p_B}$]-uniform $\mathcal{C}$ circuit. Here $\mathsf{p_B}$ stands for a set of unary predicates that allow to check the bounds on the circuit. This construction requires some minimal closure properties for the function for the functions giving the time and space bounds on the

[2] For the A_5 quantifier we do not have to distinguish between quantifiers over one variable and quantifiers over a tuple of variables.

machine. The idea is to take the machine that evaluates a circuit in $\mathcal{M}$-uniform $\mathcal{C}$ and to construct the FO$[<, \mathsf{p_B}]$-uniform circuit for this machine. An example is polynomial time where P-uniform polynomial size circuits equal FO$[<]$-uniform polynomial size circuits. (Here, $\mathsf{p_B}$ can be directly expressed within FO$[<]$.)

Similar observations hold for alternating Turing machines as exhibited in [Ruz81]: for $k > 1$ the different notions of uniformity define identical circuit families including LOGSPACE-uniform families. Then, it is easy to see how to extend the construction of Theorem 6 to circuits of depth $\log^k$.

We believe that the requirement for $\mathcal{C}$ to be contained in some Turing class can be omitted. Let M be the machine that would decide the uniformity language. The idea is the following: Let a be a gate and b be a possible candidate to be a predecessor. Instead of letting the uniformity language to decide whether there is a wire from b into a, we build a circuit, that will evaluate M and then either feed in b or not. (Note this requires to be able to feed in a neutral input.) This is done for all possible gates b for a. We call this construction a “switch gate”. So we need to be able to simulate M in the circuit class $\mathcal{C}$. This idea is can be already found in [Ruz81].

This is an explanation why log-depth circuits seem to be always at least DLOGTIME-uniform as exhibited in Section 3.

6 Binary Encoding

In this section only, we replace *unary* by *binary* in our shuffled encodings and consider the effect on the uniformity notions that arise. The *binary n-encoding* of a number $0 \leq i < 2^n$ is defined as $w_1 \cdots w_n \in \{0,1\}^n$ such that $i = \sum_j 2^{j-1} w_j$. Note that the resulting encoding differs from the encoding in [BIS90] not only in the shuffling, but in the fact that here y is also given in binary:

Definition 7 (Binary shuffled encoding). *The* binary shuffled n-encoding *of a sequence $\alpha_1, \ldots, \alpha_k$ of numbers between* 1 *and n is the shuffle of the sequence of binary n-encodings of $\alpha_1, \ldots, \alpha_k$. When n is understood, this shuffle is denoted $\langle \alpha_1, \ldots, \alpha_k \rangle_b$ (and is a word of length* $\lceil \log(n+1) \rceil$*).*

We say that a circuit family $(C_n)_{n \geq 1}$ is FO$[X]$-uniform$_{\text{bin}}$ if the language formed by the union, over all n, of the language of binary shuffled n-encodings of the tuples in L_{C_n} can be described by an FO$[X]$ formula.

Note that Theorem 5 also holds for binary encoding. It is easy to see that we can switch from unary to binary shuffled encoding. The tests for the fixed cases are clearly in FO$[+1]$, for the other tests the formula must compute ± 1 on binary numbers, but this can be done in FO$[+1]$.

For binary encoding we get a similar result for NC^1:

Theorem 6. *The classes* FO$[<, \mathsf{x2}]$-uniform NC^1 *and* FO$[+1]$-uniform$_{\text{bin}}$ NC^1 *each contain* DLOGTIME-*uniform* NC^1*.*

The proof of Theorem 6 builds the circuit for an ALOGTIME machine from its configuration tree. Both theorems exploit the locality of a computational step of a Turing machine.

7 Discussion

Our focus in this paper was the following reasoning: if circuit classes such as $\mathrm{ACC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1$ resist separation attempts, then why not tighten their uniformity and draw intuition from comparing the restricted classes?

Our results slightly extend [BL06]. They mostly concern FO[<]-uniformity, a notion provably tighter than the robust DLOGTIME-uniformity commonly accepted as the natural choice for defining fundamental circuit subclasses of P.

Our first conclusion is that *no* intuition comes out of applying to logarithmic depth circuits the tight uniformities considered here, because these circuits either retain their full power or they cannot exploit their depth beyond a constant.

Another observation is that a language (such as A_5 here) complete for a large class (such as ALOGTIME here) under the reduction relevant to a robust uniformity (such as DLOGTIME-uniformity here) may no longer be complete for the same class defined under a tighter uniformity. Indeed, here we noted that in the FO[<]-uniform world, the AC^0-closure of A_5 no longer contains TC^0.

By contrast with A_5, we observed that the $\mathbb{D}_+$ variant of the NC^1-complete formula value problem behaves differently. We could show that in the FO[<]-uniform world, the AC^0 closure of $\mathbb{D}_+$ equals ALOGTIME and thus contains TC^0. This suggested considering $\mathrm{AC}^0(\mathbb{D}_+)_{LIN} \supseteq \mathrm{TC}^0_{LIN}$ because imposing a linear bound on the fanin of unbounded fanin circuits nicely fits into the first-order characterizations of the language classes captured. In the FO[<]-uniform world (a world free of the "tyranny" of the bit predicate), can $\mathrm{AC}^0(\mathbb{D}_+)_{LIN} \neq \mathrm{TC}^0_{LIN}$ be proved? Such a separation would further amount to distinguishing the power of MAJ from the power of $\mathbb{D}_+$. We have been unable to answer that question though it might be within reach.

In [MTV10], the regularity conjecture (see footnote in Section 1) was generalized and named the "uniform duality property". Simplifying somewhat, the property holds for a class C if any language in C expressed in ExtFO[arb] can be reexpressed in $\mathrm{ExtFO}[<, C^N]$, where C^N is a set of numerical predicates defined from C. A marginal link with the uniform duality property can be found in our Theorem 5. Let ExtFO locally here mean allowing a Lindström quantifier for a P-complete problem under AC^0-reducibility. Then

$$\mathrm{ExtFO}[\mathrm{P}^N] \cap C = \mathrm{ExtFO}[+1] \cap C \subseteq \mathrm{ExtFO}[<,+] \cap C \subseteq \mathrm{ExtFO}[\mathrm{CFL}^N] \cap C$$

where C is the class of context-free languages, the "=" uses Theorem 5 and the rightmost "$\subseteq$" follows by [MTV10]. This is a weaker instance of the duality property in which we replace predicates from P^N (rather than from arb) in order to reexpress any context-free language. Can stronger instances of the duality be proven, by extending the present work or by bringing in the extensions to [RS07] recently announced in [KS12]?

The fact that we can find strictly uniform circuit classes for ALOGTIME is based on the exploitation of uniformity and locality of the steps of a Turing machine. This also allows FO[<]-uniform characterizations of polynomial time as observed in [BL06]. We think that this could be extended to circuit classes that have characterizations in terms of Turing machines, or to circuit classes whose uniformity languages are defined by Turing machines as long as the circuit class is at least as powerful as the uniformity language.

Perhaps one other research avenue would be to consider direct connection language encodings that are intermediate between the unary shuffled encoding and the binary shuffled encodings studied here. Or could a meaningful encoding-free notion of uniformity be developed? Would there be a use for such a notion?

Acknowledgement. We thank anonymous referees for useful comments on earlier versions of the present paper.

References

[All89] Allender, E.: P-uniform circuit complexity. J. ACM 36(4), 912–928 (1989)

[Bar89] Mix Barrington, D.A.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. J. Comput. Syst. Sci. 38(1), 150–164 (1989)

[BCGR92] Buss, S.R., Cook, S., Gupta, A., Ramachandran, V.: An optimal parallel algorithm for formula evaluation. SIAM J. Comput. 21(4), 755–780 (1992)

[BIS90] Mix Barrington, D.A., Immerman, N., Straubing, H.: On Uniformity within NC^1. J. Comput. Syst. Sci. 41(3), 274–306 (1990)

[BL06] Behle, C., Lange, K.-J.: FO[<]-Uniformity. In: IEEE Conference on Computational Complexity, pp. 183–189 (2006)

[Bor77] Borodin, A.: On relating time and space to size and depth. SIAM J. Comput. 6(4), 733–744 (1977)

[Bus87] Buss, S.R.: The boolean formula value problem is in alogtime. In: STOC, pp. 123–131. ACM (1987)

[Coo79] Cook, S.A.: Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In: STOC, pp. 338–345. ACM (1979)

[Gol78] Goldschlager, L.M.: A unified approach to models of synchronous parallel machines. In: STOC, pp. 89–94 (1978)

[Imm87] Immerman, N.: Languages that capture complexity classes. SIAM J. Comput. 16(4), 760–778 (1987)

[KS12] Krebs, A., Sreejith, V.: Non-definability of languages by generalized first-order formulas over $(\mathbb{N},+)$. In: LICS (to appear, 2012)

[LMSV01] Lautemann, C., McKenzie, P., Schwentick, T., Vollmer, H.: The descriptive complexity approach to LOGCFL. J. Comput. Syst. Sci. 62(4), 629–652 (2001)

[MP71] McNaughton, R., Papert, S.: Counter-free automata. With an appendix by William Henneman. In: Research Monograph No. 65, vol. XIX, 163 p. The M. I. T. Press, Cambridge (1971)

[MTV10] McKenzie, P., Thomas, M., Vollmer, H.: Extensional uniformity for boolean circuits. SIAM J. Comput. 39(7), 3186–3206 (2010)

[RS07] Roy, A., Straubing, H.: Definability of languages by generalized first-order formulas over N^+. SIAM J. Comput. 37(2), 502–521 (2007)

[Ruz81] Ruzzo, W.L.: On uniform circuit complexity. J. Comput. Syst. Sci. 22(3), 365–383 (1981)

[Str94] Straubing, H.: Finite Automata, Formal Logic, and Circuit Complexity. Birkhäuser, Boston (1994)

[Tho78] Thomas, W.: The theory of successor with an extra predicate. Math. Annalen 237, 121–132 (1978)

[Vol99] Vollmer, H.: Introduction to Circuit Complexity. Springer (1999)

The Join Levels of the Trotter-Weil Hierarchy Are Decidable

Manfred Kufleitner* and Alexander Lauser*

FMI, University of Stuttgart, Germany
{kufleitner, lauser}@fmi.uni-stuttgart.de

Abstract. The variety **DA** of finite monoids has a huge number of different characterizations, ranging from two-variable first-order logic FO^2 to unambiguous polynomials. In order to study the structure of the subvarieties of **DA**, Trotter and Weil considered the intersection of varieties of finite monoids with bands, *i.e.*, with idempotent monoids. The varieties of idempotent monoids are very well understood and fully classified. Trotter and Weil showed that for every band variety **V** there exists a unique maximal variety **W** inside **DA** such that the intersection with bands yields the given band variety **V**. These maximal varieties **W** define the Trotter-Weil hierarchy. This hierarchy is infinite and it exhausts **DA**; induced by band varieties, it naturally has a zigzag shape. In their paper, Trotter and Weil have shown that the corners and the intersection levels of this hierarchy are decidable.

In this paper, we give a single identity of omega-terms for every join level of the Trotter-Weil hierarchy; this yields decidability. Moreover, we show that the join levels and the subsequent intersection levels do not coincide. Almeida and Azevedo have shown that the join of $\mathcal{R}$-trivial and $\mathcal{L}$-trivial finite monoids is decidable; this is the first non-trivial join level of the Trotter-Weil hierarchy. We extend this result to the other join levels of the Trotter-Weil hierarchy. At the end of the paper, we give two applications. First, we show that the hierarchy of deterministic and codeterministic products is decidable. And second, we show that the direction alternation depth of unambiguous interval logic is decidable.

1 Introduction

The lattice of band varieties was classified independently by Birjukov, Fennemore, and Gerhard [3,6,7]. For the purpose of this paper, a band is a finite idempotent monoid; and a variety is a class of finite monoids which is closed under submonoids, homomorphic images, and finite direct products. We denote the variety of all bands by **B**. The relation between the band varieties can be found on the left-hand side of Figure 1 where we use the notation of [8,21]. A famous supervariety of **B** is **DA**, the class of all finite monoids such that every regular $\mathcal{D}$-class is an aperiodic semigroup. This variety appears at a huge number

* The authors acknowledge the support by the German Research Foundation (DFG) under grant DI 435/5-1.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 603–614, 2012.

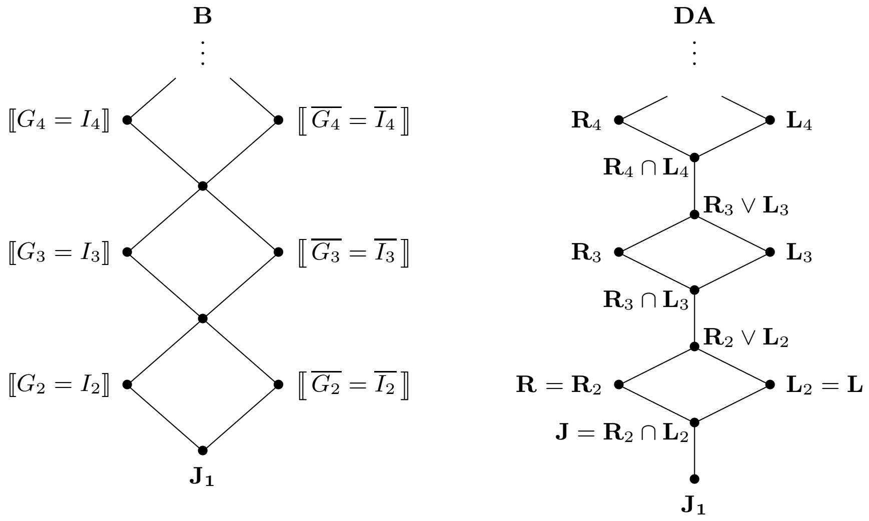

$\mathbf{B}$
$[\![G_4 = I_4]\!]$
$[\![\overline{G_4} = \overline{I_4}]\!]$
$[\![G_3 = I_3]\!]$
$[\![\overline{G_3} = \overline{I_3}]\!]$
$[\![G_2 = I_2]\!]$
$[\![\overline{G_2} = \overline{I_2}]\!]$
$\mathbf{J_1}$
$\mathbf{DA}$
$\mathbf{R}_4$
$\mathbf{L}_4$
$\mathbf{R}_4 \cap \mathbf{L}_4$
$\mathbf{R}_3 \vee \mathbf{L}_3$
$\mathbf{R}_3$
$\mathbf{L}_3$
$\mathbf{R}_3 \cap \mathbf{L}_3$
$\mathbf{R}_2 \vee \mathbf{L}_2$
$\mathbf{R} = \mathbf{R}_2$
$\mathbf{L}_2 = \mathbf{L}$
$\mathbf{J} = \mathbf{R}_2 \cap \mathbf{L}_2$
$\mathbf{J_1}$

Fig. 1. The band hierarchy and the Trotter-Weil hierarchy

of different occasions, see *e.g.* [19,4] for overviews. The most prominent results along this line of work are the following: A language is definable in two-variable first-order logic FO^2 if and only if its syntactic monoid is in $\mathbf{DA}$ [20] if and only if it is a disjoint union of unambiguous monomials [16].

Trotter and Weil studied the lattice of all subvarieties of $\mathbf{DA}$. This lattice is uncountably infinite whereas the lattice of band varieties is countably infinite. Considering the bands inside the subvarieties of $\mathbf{DA}$ led to the following result. For every given band variety $\mathbf{V}$ they showed that there exists a unique maximal variety $\mathbf{W} \subseteq \mathbf{DA}$ such that $\mathbf{V} = \mathbf{W} \cap \mathbf{B}$, *cf.* [21]. These maximal varieties $\mathbf{W}$ define the Trotter-Weil hierarchy. Its structure is depicted on the right-hand side of Figure 1. The zigzag shape gives rise to the following notions. We say the varieties $\mathbf{R}_m$ and $\mathbf{L}_m$ are the *corners*, varieties of form $\mathbf{R}_m \cap \mathbf{L}_m$ are the *intersection levels*, and $\mathbf{R}_m \vee \mathbf{L}_m$ are the *join levels* of the Trotter-Weil hierarchy. Only the corners and the intersection levels are maximal subvarieties of $\mathbf{DA}$ such that their intersection with bands yields a given band variety; by Theorem 2 below, the join levels are not maximal with this property. Kufleitner and Weil showed that there exist several different ways of climbing up along the corners of the Trotter-Weil hierarchy. One possibility is in terms of Mal'cev products with definite and reverse definite semigroups [10]; and another possibility uses condensed rankers [12]. The concept of condensed rankers is a refinement of the rankers of Weis and Immerman [22] and the turtle programs of Schwentick, Thérien, and Vollmer [17]. Condensed rankers are very similar to the unambiguous interval logic of Lodaya, Pandya, and Shah [13], which in turn gives yet another way of climbing up the Trotter-Weil hierarchy.

Kufleitner and Weil showed that the FO^2 quantifier alternation hierarchy and the Trotter-Weil hierarchy are interwoven [12]. Only recently, they tightened this connection: A language is definable in FO^2 with m blocks of quantifiers if and only if it is recognizable by a monoid in $\mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$, *cf.* [11]. Therefore, a result of Straubing on the FO^2 alternation hierarchy shows that it is possible to climb up the Trotter-Weil hierarchy along the intersection levels by using weakly iterated block products of $\mathcal{J}$-trivial monoids [18].

Pin showed that the algebraic operations of taking Mal'cev products with definite and reverse definite semigroups admit language counterparts by means of deterministic and codeterministic products [14], see also [15,16]. Thus a language over the alphabet A is recognizable by a monoid in $\mathbf{DA}$ if and only if it is in the closure of languages B^* for $B \subseteq A$ under Boolean operations and deterministic and codeterministic products [12]. This naturally defines a hierarchy of languages inside $\mathbf{DA}$. Let $\mathcal{W}_1$ be the Boolean closure of languages of the form B^*, and let $\mathcal{W}_{m+1}$ be the Boolean closure of deterministic and of codeterministic products of languages in $\mathcal{W}_m$. Now, a language is in $\mathcal{W}_m$ if and only if it is recognizable by a monoid in $\mathbf{R}_m \vee \mathbf{L}_m$. In particular, this is an infinite hierarchy which exhausts the class of all $\mathbf{DA}$-recognizable languages. Note that if one would replace deterministic and codeterministic products by unambiguous products, then Schützenberger's result [16] shows that the resulting hierarchy collapses at level 2.

Our main result Theorem 1 gives a single identity of omega-terms for each of the varieties $\mathbf{R}_m \vee \mathbf{L}_m$. It follows that membership in $\mathbf{R}_m \vee \mathbf{L}_m$ is decidable. Since $\mathbf{R}_2$ is the class of $\mathcal{R}$-trivial monoids and $\mathbf{L}_2$ is the class of $\mathcal{L}$-trivial monoids, this extends the decidability result for the join of $\mathcal{R}$-trivial and $\mathcal{L}$-trivial monoids by Almeida and Azevedo [2] to the other join levels of the Trotter-Weil hierarchy. In fact, the Almeida-Azevedo result is the base of our proof. As a byproduct, we give a new single identity of omega-terms for the corners of the Trotter-Weil hierarchy. Different identities were obtained by Trotter and Weil [21]. We complement our main result by showing that, for every $m \geq 2$, the variety $\mathbf{R}_m \vee \mathbf{L}_m$ is strictly contained in $\mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$, see Theorem 2. Note that for band varieties, the join levels coincide with the subsequent intersection levels, see *e.g.* [8].

We give two applications. The first one is decidability of the hierarchy $\mathcal{W}_m$ of deterministic and codeterministic products, see Proposition 4. This easily follows from our main result and from Pin's characterization of deterministic and codeterministic products [14]. The second application (Corollary 3) is the following: For every integer m it is decidable whether a given regular language L is definable in unambiguous interval logic with at most m direction alternations, see Section 6.2 for definitions.

2 Preliminaries

Words and Languages. Throughout this paper we let A be a finite alphabet. The set of finite words over A is denoted by A^*. It is the free monoid over A. The *empty word* 1 is the neutral element. As usual, we set $A^+ = A^* \setminus \{1\}$. The *length* of a word $u = a_1 \cdots a_n$ with $a_i \in A$ is $|u| = n$, and its *alphabet* (also known as its *content*) is the set $\alpha(u) = \{a_1, \ldots, a_n\}$. A homomorphism

$\varphi : A^* \to M$ to a monoid M *recognizes* a language $L \subseteq A^*$ if $\varphi^{-1}(\varphi(L)) = L$. A monoid M *recognizes* $L \subseteq A^*$ if there exists a homomorphism $\varphi : A^* \to M$ which recognizes L. Every language admits a unique minimal monoid M which recognizes L. This monoid is called the *syntactic monoid* of L, see *e.g.* [15] for details. A language L is *regular* (or *recognizable*) if it is recognized by a finite monoid.

Finite Monoids. Let M be a monoid. An element $e \in M$ is *idempotent* if $e^2 = e$. If M is finite, then there exists a positive integer ω such that x^ω is the unique idempotent element generated by $x \in M$. *Green's relations* $\mathcal{R}$ and $\mathcal{L}$ are an important tool for describing the structure of monoids. For $x, y \in M$ we define

$$\begin{aligned} &x \mathrel{\mathcal{R}} y \text{ if and only if } xM = yM, && x \leq_{\mathcal{R}} y \text{ if and only if } xM \subseteq yM,\\ &x \mathrel{\mathcal{L}} y \text{ if and only if } Mx = My, && x \leq_{\mathcal{L}} y \text{ if and only if } Mx \subseteq My. \end{aligned}$$

We frequently use these relations as follows: The relation $x \leq_{\mathcal{R}} y$ holds if and only if there exists $z \in M$ such that $x = yz$, Similarly, $x \leq_{\mathcal{L}} y$ if and only if there exists $z \in M$ such that $x = zy$. We say that a monoid M is $\mathcal{R}$*-trivial* (resp. $\mathcal{L}$*-trivial*) if $\mathcal{R}$ (resp. $\mathcal{L}$) is the identity relation on M.

Every homomorphism $\varphi : A^* \to M$ induces a congruence $\equiv_\varphi$ on A^* by setting $x \equiv_\varphi y$ if $\varphi(x) = \varphi(y)$. Now, the submonoid $\varphi(A^*)$ of M is isomorphic to $A^*/{\equiv_\varphi}$. For defining the Trotter-Weil hierarchy, we introduce the congruences $\sim_K$ and $\sim_D$ on M. We let $x \sim_K y$ if for all idempotents e of M we have:

$$\text{if } ex \mathrel{\mathcal{R}} e \text{ or } ey \mathrel{\mathcal{R}} e, \text{ then } ex = ey.$$

Using more semigroup theoretic notions, the meaning of $x \sim_K y$ is that, for every regular $\mathcal{D}$-class D, the right translations by x and by y define the same partial function on D, see *e.g.* [9]. The left-right dual $\sim_D$ is defined by $x \sim_D y$ if for all idempotents e of M we have that if $xe \mathrel{\mathcal{L}} e$ or $ye \mathrel{\mathcal{L}} e$, then $xe = ye$.

Varieties of Finite Monoids. A *variety* is a class of finite monoids which is closed under submonoids, homomorphic images, and finite direct products. The empty direct product of monoids yields the one-element monoid $\{1\}$. Thus the monoid $\{1\}$ is contained in every variety. The *join* $\mathbf{V} \vee \mathbf{W}$ of two varieties $\mathbf{V}$ and $\mathbf{W}$ is the smallest variety containing both $\mathbf{V}$ and $\mathbf{W}$. A *language variety* is a class of regular languages which is closed under Boolean operations, inverse homomorphic images, and residuals. More formally, the languages in a language variety are parametrized by the alphabet, but in order to keep the notation in this paper simple, we use this distinction only implicitly. Eilenberg has shown that there is a one-to-one correspondence between language varieties and varieties of finite monoids [5]. This connection is defined by the following mutually inverse correspondences: To every language variety $\mathcal{V}$ one assigns the variety of finite monoids generated by the syntactic monoids of languages in $\mathcal{V}$; and to every variety $\mathbf{V}$ of finite monoids one assigns the languages recognized by the monoids in $\mathbf{V}$.

Identities of omega-terms are a very common way of defining varieties. We inductively define *omega-terms* over a set of variables Σ. The empty word 1 and every $x \in \Sigma$ is an omega-term; and if u and v are omega-terms, then so are uv and $(u)^\omega$. Every homomorphism $\varphi : \Sigma^* \to M$ to a finite monoid M naturally extends to omega-terms by setting $\varphi(u^\omega) = \varphi(u)^\omega$, *i.e.*, $\varphi(u^\omega)$ is the idempotent generated by $\varphi(u)$. Let u and v be two omega-terms over Σ. A finite monoid M *satisfies* the identity $u = v$ if for every homomorphism $\varphi : \Sigma^* \to M$ we have $\varphi(u) = \varphi(v)$. The class of all finite monoids which satisfy $u = v$ is denoted by $[\![u = v]\!]$. This class is a variety for all omega-terms u, v. The monoids in $[\![x^\omega = x^\omega x]\!]$ are called *aperiodic*. We will use the following varieties in this paper:

$$\begin{aligned}
\mathbf{DA} &= [\![(xy)^\omega x (xy)^\omega = (xy)^\omega]\!]\\
\mathbf{R} &= [\![(zx)^\omega z = (zx)^\omega]\!]\\
\mathbf{L} &= [\![z(yz)^\omega = (yz)^\omega]\!]\\
\mathbf{J} &= \mathbf{R} \cap \mathbf{L}\\
\mathbf{B} &= [\![x^2 = x]\!]\\
\mathbf{J_1} &= [\![xy = yx]\!] \cap \mathbf{B}\\
\mathbf{W}_m &= [\![e_m \cdots e_1 z f_1 \cdots f_m = e_m \cdots e_1 f_1 \cdots f_m]\!] \text{ where}\\
&\qquad e_1 = 1 \text{ and } e_{i+1} = (e_i \cdots e_1 z f_1 \cdots f_i x_i)^\omega,\\
&\qquad f_1 = 1 \text{ and } f_{i+1} = (y_i e_i \cdots e_1 z f_1 \cdots f_i)^\omega.
\end{aligned}$$

The identity for $\mathbf{W}_m$ uses variables $x_1, \ldots, x_{m-1}$ as well as $y_1, \ldots, y_{m-1}$ and z. In particular, we have $\mathbf{W}_2 = [\![(zx_1)^\omega z (y_1 z)^\omega = (zx_1)^\omega (y_1 z)^\omega]\!]$. Using $x_i = y_i = 1$ for all $1 \leq i < m$ we see that all monoids in $\mathbf{W}_m$ are aperiodic. A monoid is in $\mathbf{R}$ if and only if it is $\mathcal{R}$-trivial, and it is in $\mathbf{L}$ if and only if it is $\mathcal{L}$-trivial, see *e.g.* [15]. The elements of $\mathbf{J}$ are called $\mathcal{J}$*-trivial* monoids.

If $\mathbf{V}$ is a variety, then M is in the variety $\mathbf{K} ⓜ \mathbf{V}$ if $M/{\sim_K}$ is in $\mathbf{V}$. Symmetrically, M is in the variety $\mathbf{D} ⓜ \mathbf{V}$ if $M/{\sim_D}$ is in $\mathbf{V}$. Usually, the Mal'cev products $\mathbf{K} ⓜ \mathbf{V}$ and $\mathbf{D} ⓜ \mathbf{V}$ are defined using relational morphisms, but the definition given here is equivalent [9]. We are now ready to define the *Trotter-Weil hierarchy*. We set $\mathbf{R}_2 = \mathbf{R}$ and $\mathbf{L}_2 = \mathbf{L}$, and for $m \geq 2$ we let $\mathbf{R}_{m+1} = \mathbf{K} ⓜ \mathbf{L}_m$ and $\mathbf{L}_{m+1} = \mathbf{D} ⓜ \mathbf{R}_m$. There are several possible extensions to the first level so as to obtain the same hierarchy for the higher levels: we have $\mathbf{K} ⓜ \mathbf{V} = \mathbf{R}_2$ and $\mathbf{D} ⓜ \mathbf{V} = \mathbf{L}_2$ for every variety $\mathbf{V}$ with $\mathbf{J_1} \subseteq \mathbf{V} \subseteq \mathbf{J}$, see *e.g.* [15]. It depends on the context what the most natural choice is, and we therefore start the hierarchy at level 2. The structure of the Trotter-Weil hierarchy is depicted on the right-hand side of Figure 1. It is well-known that $\bigcup_m \mathbf{R}_m = \bigcup_m \mathbf{L}_m = \mathbf{DA}$, see [10]. We also note that every m-generated finite monoid in $\mathbf{DA}$ is in $\mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$, see *e.g.* [12, Proposition 3.23]. The main purpose of this paper is to show

$$\mathbf{W}_m = \mathbf{R}_m \vee \mathbf{L}_m \subsetneq \mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}.$$

The containment $\mathbf{R}_m \vee \mathbf{L}_m \subseteq \mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$ is straightforward, see [12, Corollary 3.19]. Almeida and Azevedo [2] have shown that $\mathbf{R} \vee \mathbf{L} = \mathbf{W}_2$, see also [1].

3 Identities for the Corners

The following lemma is one of the main properties of monoids in $\mathbf{DA}$. It basically says that, inside $\mathbf{DA}$, whether or not $u \mathrel{\mathcal{R}} ua$ only depends on a and the $\mathcal{R}$-class of u, but not on the element u itself. Symmetrically, for monoids in $\mathbf{DA}$, whether or not $u \mathrel{\mathcal{L}} au$ only depends on a and the $\mathcal{L}$-class of u.

Lemma 1. *Let $\varphi : A^* \to M$ be a homomorphism with $M \in \mathbf{DA}$, let $u, v \in M$, and let $x, y \in A^*$ with $\alpha(x) = \alpha(y)$. If $u \mathrel{\mathcal{R}} v$, then $u \mathrel{\mathcal{R}} u\varphi(x)$ if and only if $v \mathrel{\mathcal{R}} v\varphi(y)$. If $u \mathrel{\mathcal{L}} v$, then $u \mathrel{\mathcal{L}} \varphi(x)u$ if and only if $v \mathrel{\mathcal{L}} \varphi(y)v$.*

The next lemma shows that we can apply Lemma 1 for $M \in \mathbf{W}_m$.

Lemma 2. *For all $m \geq 2$ we have $\mathbf{W}_m \subseteq \mathbf{DA}$.*

Proof. Let $M \in \mathbf{W}_m$. Setting $x_i = y_i = z$ for all i yields $e_i = f_i = z^\omega$ and consequently $z^\omega z = z^\omega$, that is, M is aperiodic. With $x_i = y_i = y$ for all i we get $e_i = (zy)^\omega$ and $f_i = (yz)^\omega$. The defining identity for $\mathbf{W}_m$ implies $(zy)^\omega(yz)^\omega = (zy)^\omega z$. Hence, $(yz)^\omega y (yz)^\omega = y(zy)^\omega(yz)^\omega = y(zy)^\omega z = (yz)^\omega$. The last equality relies on the aperiodicity of M. □

The next proposition gives a new equational description of the corners of the Trotter-Weil hierarchy. This description immediately yields $\mathbf{R}_m \vee \mathbf{L}_m \subseteq \mathbf{W}_m$.

Proposition 1. *Let $e_1 = f_1 = 1$ and for $i \geq 1$ let $e_{i+1} = (e_i \cdots e_1 z f_1 \cdots f_i x_i)^\omega$ and $f_{i+1} = (y_i e_i \cdots e_1 z f_1 \cdots f_i)^\omega$. Then for all $m \geq 2$ we have*

$$\mathbf{R}_m = \llbracket e_m \cdots e_1 z f_1 \cdots f_{m-1} = e_m \cdots e_1 f_1 \cdots f_{m-1} \rrbracket \,,$$
$$\mathbf{L}_m = \llbracket e_{m-1} \cdots e_1 z f_1 \cdots f_m = e_{m-1} \cdots e_1 f_1 \cdots f_m \rrbracket \,.$$

Proof. For $m = 2$ the claim is true by definition of $\mathbf{R}_2$ and $\mathbf{L}_2$. Let now $m \geq 3$. By left-right symmetry it suffices to show the statement for $\mathbf{R}_m$. First, we consider the inclusion from left to right. Let $M \in \mathbf{R}_m$. Then $M/{\sim_K} \in \mathbf{L}_{m-1}$ and by induction, $M/{\sim_K} \in \llbracket e_{m-2} \cdots e_1 z f_1 \cdots f_{m-1} = e_{m-2} \cdots e_1 f_1 \cdots f_{m-1} \rrbracket$, *i.e.*, the elements $u = e_{m-2} \cdots e_1 z f_1 \cdots f_{m-1}$ and $v = e_{m-2} \cdots e_1 f_1 \cdots f_{m-1}$ satisfy $u \sim_K v$ in M. Thus $e_{m-1}u \sim_K e_{m-1}v$. We have $e_m \mathrel{\mathcal{R}} e_m e_{m-1} u$ because $e_m \leq_{\mathcal{R}} e_{m-1}u$. Hence, $e_m e_{m-1} u = e_m e_{m-1} v$ by definition of $\sim_K$.

Next, we show the inclusion from right to left. Let M satisfy the identity $e_m \cdots e_1 z f_1 \cdots f_{m-1} = e_m \cdots e_1 f_1 \cdots f_{m-1}$. We have $M \in \mathbf{DA}$ by Lemma 2. For $u = e_{m-2} \cdots e_1 z f_1 \cdots f_{m-1}$ and $v = e_{m-2} \cdots e_1 f_1 \cdots f_{m-1}$ we claim $u \sim_K v$. Then $M/{\sim_K} \in \mathbf{L}_{m-1}$ by induction which yields $M \in \mathbf{R}_m$. Consider an idempotent element $e \in M$ such that $eu \mathrel{\mathcal{R}} e$. Note that by Lemma 1, we have $eu \mathrel{\mathcal{R}} e$ if and only if $ev \mathrel{\mathcal{R}} e$. There exists $x_{m-1} \in M$ with $e = eux_{m-1}$. Let $x_{m-2} = f_{m-1}x_{m-1}$, let $e_{m-1} = (e_{m-2} \cdots e_1 z f_1 \cdots f_{m-2} x_{m-2})^\omega$, and let $e_m = (e_{m-1} e_{m-2} \cdots e_1 z f_1 \cdots f_{m-1} x_{m-1})^\omega$. One can verify that $e_m = e_{m-1} = (ux_{m-1})^\omega$, from which it follows that $e = ee_m e_{m-1}$. By the choice of M we have $e_m e_{m-1} u = e_m e_{m-1} v$ and hence, $eu = ee_m e_{m-1} u = ee_m e_{m-1} v = ev$. This shows $u \sim_K v$. □

4 Rankers

Condensed rankers are important for the proofs of both of our main results Theorem 1 and Theorem 2. A *ranker* is a nonempty word over the alphabet $Z_A = \{\mathsf{X}_a, \mathsf{Y}_a \mid a \in A\}$ with $2\,|A|$ letters. The set Z_A is partitioned into $Z_A = X_A \cup Y_A$ with $X_A = \{\mathsf{X}_a \mid a \in A\}$ and $Y_A = \{\mathsf{Y}_a \mid a \in A\}$. Every ranker is interpreted as a sequence of instructions of the form "go to the next a-position" and "go to the previous a-position". More formally, for $u = a_1 \cdots a_n \in A^*$ and $x \in \{0, \ldots, n+1\}$ we let

$$\begin{aligned} \mathsf{X}_a(u,x) &= \min\{y \mid y > x \text{ and } a_y = a\}, & \mathsf{X}_a(u) &= \mathsf{X}_a(u,0), \\ \mathsf{Y}_a(u,x) &= \max\{y \mid y < x \text{ and } a_y = a\}, & \mathsf{Y}_a(u) &= \mathsf{Y}_a(u,n+1). \end{aligned}$$

Here, both the minimum and the maximum of the empty set are undefined. The modality X_a is for "neXt-a" and Y_a is for "Yesterday-a". For a ranker $r = \mathsf{Z}s$ with $\mathsf{Z} \in Z_A$ we set $r(u) = s(u, \mathsf{Z}(u))$ and $r(u,x) = s(u, \mathsf{Z}(u,x))$. In particular, the instructions of a ranker are executed from left to right. Every ranker r either defines a unique position in a word u, or it is undefined on u. For example, $\mathsf{X}_a\mathsf{Y}_b\mathsf{X}_c(bca) = 2$ and $\mathsf{X}_a\mathsf{Y}_b\mathsf{X}_c(bac) = 3$ whereas $\mathsf{X}_a\mathsf{Y}_b\mathsf{X}_c(abac)$ and $\mathsf{X}_a\mathsf{Y}_b\mathsf{X}_c(bcba)$ are undefined. A ranker r is *condensed* on u if it is defined and, during the execution of r, no previously visited position is overrun [12]. More formally, let $r = \mathsf{Z}_1 \cdots \mathsf{Z}_k$ with $\mathsf{Z}_i \in Z_A$ be defined on u and let $x_i = \mathsf{Z}_1 \cdots \mathsf{Z}_i(u)$ be the position reached after i instructions. Then r is *condensed* on u if for every $i \leq k-1$ we have that either all positions $x_{i+1}, \ldots, x_k$ are greater than x_i or all positions $x_{i+1}, \ldots, x_k$ are smaller than x_i. By definition, every ranker which is condensed on u is also defined on u, but the converse is not true. For example, $\mathsf{X}_a\mathsf{Y}_b\mathsf{X}_c$ is condensed on bca but not on bac. Let

$$L_c(r) = \{u \in A^* \mid r \text{ is condensed on } u\}.$$

The *depth* of a ranker is its length as a word. A *block* of a ranker is a maximal factor either in X_A^+ or Y_A^+. A ranker with m blocks changes direction $m-1$ times. By $R_{m,n}$ we denote the rankers with depth at most n and with up to m blocks. Let $R_m = \bigcup_n R_{m,n}$. We set $R_{m,n}^{\mathsf{X}} = (R_{m,n} \cap X_A Z_A^*) \cup R_{m-1,n-1}$ and $R_{m,n}^{\mathsf{Y}} = (R_{m,n} \cap Y_A Z_A^*) \cup R_{m-1,n-1}$. We write $u \vartriangleright_{m,n} v$ (resp. $u \vartriangleleft_{m,n} v$) if u and v are condensed on the same rankers in $R_{m,n}^{\mathsf{X}}$ (resp. $R_{m,n}^{\mathsf{Y}}$). Let $u \equiv_{m,n} v$ if u and v are condensed on the same rankers from $R_{m,n}$, *i.e.*, if both $u \vartriangleright_{m,n} v$ and $u \vartriangleleft_{m,n} v$. The relations $\vartriangleright_{m,n}$ and $\vartriangleleft_{m,n}$ are finite index congruences [12, Lemma 3.13] and hence so is $\equiv_{m,n}$.

The following result of Kufleitner and Weil shows that condensed rankers can be used for defining the languages corresponding to the Trotter-Weil hierarchy [12, Theorem 3.21].

Proposition 2. *For all $m, n \in \mathbb{N}$ we have $A^*/{\vartriangleright_{m,n}} \in \mathbf{R}_m$ and $A^*/{\vartriangleleft_{m,n}} \in \mathbf{L}_m$. For every homomorphism $\varphi : A^* \to M$ with $M \in \mathbf{R}_m$ (resp. $M \in \mathbf{L}_m$) there exists $n \in \mathbb{N}$ such that $u \vartriangleright_{m,n} v$ (resp. $u \vartriangleleft_{m,n} v$) implies $\varphi(u) = \varphi(v)$ for all $u, v \in A^*$.*

This leads to the following corollary, which is the main motivation for considering condensed rankers in this paper.

Corollary 1. *For all $m, n \in \mathbb{N}$ we have $A^*/{\equiv_{m,n}} \in \mathbf{R}_m \vee \mathbf{L}_m$. For every homomorphism $\varphi : A^* \to M$ with $M \in \mathbf{R}_m \vee \mathbf{L}_m$ there exists $n \in \mathbb{N}$ such that $u \equiv_{m,n} v$ implies $\varphi(u) = \varphi(v)$ for all $u, v \in A^*$.*

5 Identities for the Join Levels

This section contains our main contribution. Theorem 1 gives a description of the join levels of the Trotter-Weil hierarchy by a single identity of omega-terms. Since for every given monoid M one can effectively verify whether or not M satisfies this identity, Theorem 1 immediately yields Corollary 2.

Theorem 1. *For every $m \geq 2$ we have $\mathbf{R}_m \vee \mathbf{L}_m = \mathbf{W}_m$.*

Corollary 2. *For every given integer $m \geq 2$ and every given finite monoid M it is decidable whether M is in $\mathbf{R}_m \vee \mathbf{L}_m$.*

The proof of the inclusion $\mathbf{W}_m \subseteq \mathbf{R}_m \vee \mathbf{L}_m$ in Theorem 1 is by induction on m. The base case $m = 2$ was shown by Almeida and Azevedo [2]. The induction step connects condensed rankers with the variety $\mathbf{W}_m$. It uses rewriting techniques (see Proposition 3) and relies on a combination of properties of $\mathbf{DA}$ and condensed rankers (given in Lemma 3). Corollary 1 yields the connection between condensed rankers and the join levels of the Trotter-Weil hierarchy.

Lemma 3. *Let $m \geq 3$, let $n \in \mathbb{N}$, and let $\varphi : A^* \to M$ be a homomorphism with $M \in \mathbf{DA}$. Suppose for all $u, v, x, y \in A^*$ the following implication holds:*

$$u \equiv_{m-1,n} v, \;\; \varphi(x) \mathrel{\mathcal{R}} \varphi(xu), \;\; \varphi(y) \mathrel{\mathcal{L}} \varphi(vy) \;\; \Rightarrow \;\; xuy \equiv_\varphi xvy.$$

Then $u \equiv_{m,n+2|M|-1} v$ implies $u \equiv_\varphi v$ for all $u, v \in A^*$.

Proposition 3. *Let $m \geq 2$ and let $\varphi : A^* \to M$ be a homomorphism with $M \in \mathbf{W}_m$. Then there exists $n \in \mathbb{N}$ such that $u \equiv_{m,n} v$ implies $\varphi(u) = \varphi(v)$.*

Proof. The proof is by induction on m. Almeida and Azevedo have shown $\mathbf{R}_2 \vee \mathbf{L}_2 = \mathbf{W}_2$, see [2,1]. Thus, by the second statement in Corollary 1, the claim holds for $m = 2$. Let now $m > 2$ and let $\varphi : A^* \to M$ be a surjective homomorphism onto $M \in \mathbf{W}_m$. We extend relations $\mathcal{G}$ on M to words by setting $u \mathrel{\mathcal{G}} v$ if and only if $\varphi(u) \mathrel{\mathcal{G}} \varphi(v)$. Let $\omega \geq 1$ be an integer such that x^ω is idempotent for all $x \in M$.

For words $u, v \in A^*$ we set $u \to v$ if $u \equiv_\varphi v$ or if $u = pe_{m-1} \cdots e_1 z f_1 \cdots f_{m-1} q$ and $v = pe_{m-1} \cdots e_1 f_1 \cdots f_{m-1} q$ for some words $p, q, e_i, f_i, x_i, y_i, z \in A^*$ with $e_1 = 1 = f_1$ and $e_{i+1} = (e_i \cdots e_1 z f_1 \cdots f_i x_i)^\omega$ and $f_{i+1} = (y_i e_i \cdots e_1 z f_1 \cdots f_i)^\omega$. One can think of $\to$ as a semi-Thue system induced by equality in M and the identity for $\mathbf{W}_{m-1}$. We let $\overset{*}{\leftrightarrow}$ be the equivalence relation generated by $\to$. That is, $u \overset{*}{\leftrightarrow} v$ if there exists $w_0, \ldots, w_k$ with $u = w_0$, $v = w_k$ and with $w_i \to w_{i+1}$

or $w_{i+1} \to w_i$ for all $0 \leq i < k$. The relation $\overset{*}{\leftrightarrow}$ is a congruence on A^* with finite index, and $A^*/\overset{*}{\leftrightarrow}$ is in $\mathbf{W}_{m-1}$ by definition. Note that $\varphi(u) = \varphi(v)$ implies $u \overset{*}{\leftrightarrow} v$ and therefore, $A^*/\overset{*}{\leftrightarrow}$ is a quotient of M. In particular, we have $u^{2\omega} \overset{*}{\leftrightarrow} u^{\omega}$. By induction there exists n such that $u \equiv_{m-1,n} v$ implies $u \overset{*}{\leftrightarrow} v$.

Next, we show that M satisfies the assumptions of Lemma 3. Suppose $u \equiv_{m-1,n} v$, $x \mathrel{\mathcal{R}} xu$, and $y \mathrel{\mathcal{L}} vy$. By choice of n there exists $u = w_0, \ldots, w_k = v$ with $w_i \to w_{i+1}$ or $w_{i+1} \to w_i$. We claim that if $t \to w$ and $x \mathrel{\mathcal{R}} xt$ and $y \mathrel{\mathcal{L}} ty$, then $xty \equiv_\varphi xwy$. This is trivial if $t \equiv_\varphi w$. Suppose $t = pt'q$ and $w = pw'q$ with $t' = e_{m-1} \cdots e_1 z f_1 \cdots f_{m-1}$ and $w' = e_{m-1} \cdots e_1 f_1 \cdots f_{m-1}$ where $e_1 = 1 = f_1$ and $e_{i+1} = (e_i \cdots e_1 z f_1 \cdots f_i x_i)^\omega$ and $f_{i+1} = (y_i e_i \cdots e_1 z f_1 \cdots f_i)^\omega$. We have $xp \mathrel{\mathcal{R}} xpt'$ because $x \mathrel{\mathcal{R}} xt$. Hence there exists $x_{m-1} \in A^*$ such that $xp \equiv_\varphi xpt'x_{m-1} \equiv_\varphi upe_m$ with $e_m = (t'x_{m-1})^\omega = (e_{m-1} \cdots e_1 z f_1 \cdots f_{m-1} x_{m-1})^\omega$. Symmetrically, for some $y_{m-1} \in A^*$ and $f_m = (y_{m-1} e_{m-1} \cdots e_1 z f_1 \cdots f_{m-1})^\omega$ we get $qy \equiv_\varphi f_m qy$. With $M \in \mathbf{W}_m$ we conclude

$$\begin{aligned} xty = {} & xpe_{m-1} \cdots e_1 z f_1 \cdots f_{m-1} qy \\ \equiv_\varphi {} & xpe_m e_{m-1} \cdots e_1 z f_1 \cdots f_{m-1} f_m qy \\ \equiv_\varphi {} & xpe_m e_{m-1} \cdots e_1 \;\; f_1 \cdots f_{m-1} f_m qy \\ \equiv_\varphi {} & xpe_{m-1} \cdots e_1 f_1 \cdots f_{m-1} qy = xwy. \end{aligned}$$

If $t \to w$, then either $t \equiv_\varphi w$ or $\alpha(t) = \alpha(w)$. Therefore, by Lemma 1, we have $x \mathrel{\mathcal{R}} xt$ if and only if $x \mathrel{\mathcal{R}} xw$, and $y \mathrel{\mathcal{L}} ty$ if and only if $y \mathrel{\mathcal{L}} wy$ whenever $t \to w$. Thus $x \mathrel{\mathcal{R}} xw_i$ and $y \mathrel{\mathcal{L}} w_i y$ for all $0 \leq i \leq k$. The above claim now yields $xuy = xw_0y \equiv_\varphi xw_1y \equiv_\varphi \cdots \equiv_\varphi xw_ky = xvy$. Now, by Lemma 3, we see that $u \equiv_{m,n+2|M|-1} v$ implies $u \equiv_\varphi v$. □

Proof (Theorem 1). We have $\mathbf{R}_m \subseteq \mathbf{W}_m$ and $\mathbf{L}_m \subseteq \mathbf{W}_m$ by Proposition 1. Since $\mathbf{W}_m$ is a variety, we see that $\mathbf{R}_m \vee \mathbf{L}_m \subseteq \mathbf{W}_m$. The converse inclusion follows by Proposition 3 and the first statement in Corollary 1. □

Separating the Join Levels from the Intersection Levels

For every $m \geq 2$ we show that there is a language which is recognized by a monoid in $\mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$, but not by a monoid in $\mathbf{R}_m \vee \mathbf{L}_m$. This last statement relies on Theorem 1; and the membership in $\mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$ uses condensed rankers and Proposition 2.

Theorem 2. *For all $m \geq 2$ we have $\mathbf{R}_m \vee \mathbf{L}_m \subsetneq \mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$.*

6 Applications

In this section, we relate the join levels of the Trotter-Weil hierarchy with two other hierarchies. First, we consider the hierarchy of deterministic and codeterministic products, starting with languages of the form B^* for $B \subseteq A$; and we show that this hierarchy is decidable. This result essentially follows from

Theorem 1 and from Pin's characterization of deterministic and codeterministic products [14]. Our second application is unambiguous interval temporal logic (unambiguous ITL). It has been introduced by Lodaya, Pandya, and Shah [13] as an expressively complete logic for **DA**. We show that the direction alternation depth within unambiguous ITL is decidable. This result heavily relies on combinatorial properties of rankers given by Kufleitner and Weil [12].

6.1 Deterministic and Codeterministic Products

A language of the form $L = L_0 a_1 L_1 \cdots a_k L_k$ with $L_i \in \mathcal{V}$ and $a_i \in A$ is called a *monomial* over a class of languages $\mathcal{V}$. The monomial L is *unambiguous* if every word $u \in L$ has a unique factorization $u = u_0 a_1 u_1 \cdots a_k u_k$ such that $u_i \in L_i$; it is *deterministic* if for every word $u \in L$ and every $i \in \{1, \ldots, k\}$ there is a unique prefix of u which is in $L_0 a_1 \cdots L_{i-1} a_i$; and it is *codeterministic* (also called *reverse deterministic*) if for every word $u \in L$ and every $i \in \{1, \ldots, k\}$ there is a unique suffix of u in $a_i L_i \cdots a_k L_k$. Every deterministic monomial and every codeterministic monomial is unambiguous.

If $\mathcal{V}$ is a class of languages, then the *deterministic closure* $\mathcal{V}^{det}$ of $\mathcal{V}$ (resp. the *codeterministic closure* $\mathcal{V}^{codet}$ of $\mathcal{V}$) is the Boolean closure of the deterministic (resp. codeterministic) monomials over $\mathcal{V}$. Alternating between closure under deterministic and codeterministic monomials and between closure under Boolean operations yields the following hierarchy: $\mathcal{W}_1$ contains all Boolean combinations of languages B^* for $B \subseteq A$, and $\mathcal{W}_{m+1}$ consists of all Boolean combinations of deterministic and codeterministic monomials over $\mathcal{W}_m$, *i.e.*, $\mathcal{W}_{m+1}$ is the Boolean closure of $\mathcal{W}_m^{det} \cup \mathcal{W}_m^{codet}$. Since $\mathcal{V}^{det}$ and $\mathcal{V}^{codet}$ are varieties if $\mathcal{V}$ is a variety [14], each of the language classes $\mathcal{W}_m$ is a variety. The next proposition shows that $\mathcal{W}_m$ corresponds to the level $\mathbf{R}_m \vee \mathbf{L}_m$ of the Trotter-Weil hierarchy.

Proposition 4. *Let $L \subseteq A^*$ and let $m \geq 2$. Then L is in $\mathcal{W}_m$ if and only if L is recognized by a monoid in $\mathbf{R}_m \vee \mathbf{L}_m$. In particular, membership in $\mathcal{W}_m$ is decidable.*

6.2 Alternation within Unambiguous ITL

The syntax of *unambiguous interval temporal logic* (unambiguous ITL) is as follows. Formulae are built from the atoms $\top$ for *true* and $\bot$ for *false*, and if φ and ψ are formulae in unambiguous ITL, then so are

$$\neg\varphi \;\mid\; \varphi \vee \psi \;\mid\; \varphi \wedge \psi \;\mid\; \varphi \,\mathsf{F}_a\, \psi \;\mid\; \varphi \,\mathsf{L}_a\, \psi$$

for every letter $a \in A$. The modality F_a stands for "First a-position" and L_a is for "Last a-position". Usually, models are finite words and an interval of positions. For the purpose of this paper, we use only word models (without intervals). The semantics is as follows. Every word $u \in A^*$ models $\top$, written as $u \models \top$, and no word models $\bot$. Boolean combinations are as usual. The semantics of F_a and L_a is given by

$$\begin{aligned} u \models \varphi \mathsf{F}_a \psi \;&\Leftrightarrow\; a \in \alpha(u) \text{ and for } u = u_0 a u_1 \text{ with } a \notin \alpha(u_0) \\ &\;\; \text{we have } u_0 \models \varphi \text{ and } u_1 \models \psi, \\ u \models \varphi \mathsf{L}_a \psi \;&\Leftrightarrow\; a \in \alpha(u) \text{ and for } u = u_0 a u_1 \text{ with } a \notin \alpha(u_1) \\ &\;\; \text{we have } u_0 \models \varphi \text{ and } u_1 \models \psi. \end{aligned}$$

That is, for the formula $\varphi \mathsf{F}_a \psi$, the model is "split" at the first a-position and φ and ψ are interpreted over the resulting left and right factors, respectively. The modality L_a is the left-right dual, "splitting" at the last a-position. For a formula φ we set $L(\varphi) = \{u \in A^* \mid u \models \varphi\}$. Then

$$L(\varphi \mathsf{F}_a \psi) = (L(\varphi) \cap B^*) a L(\psi), \qquad L(\varphi \mathsf{L}_a \psi) = L(\varphi) a (L(\psi) \cap B^*)$$

where $B = A \setminus \{a\}$. We introduce the parameter t for "*turns*" of a formula: we let $t(\top) = t(\bot) = 0$, $t(\neg\varphi) = t(\varphi)$ and $t(\varphi \vee \psi) = t(\varphi \wedge \psi) = \max\{t(\varphi), t(\psi)\}$; and for the temporal modalities we let

$$t(\varphi \mathsf{F}_a \psi) = \max\{t(\varphi) + 1, t(\psi)\}, \qquad t(\varphi \mathsf{L}_a \psi) = \max\{t(\varphi), t(\psi) + 1\}.$$

The parameter t defines the number of direction alternations in a formula (more precisely, the number of blocks of directions). We shall also need the following parameter $d(\varphi)$ capturing the nesting depth of F_a and L_a of the formula: let $d(\top) = d(\bot) = 0$, $d(\neg\varphi) = d(\varphi)$ and $d(\varphi \vee \psi) = d(\varphi \wedge \psi) = \max\{d(\varphi), d(\psi)\}$; and for the temporal modalities we set $d(\varphi \mathsf{F}_a \psi) = d(\varphi \mathsf{L}_a \psi) = 1 + \max\{d(\varphi), d(\psi)\}$. Let $\mathrm{ITL}_{m,n}$ contain all unambiguous ITL-formulae φ with $t(\varphi) \leq m$ and $d(\varphi) \leq n$. Let $\mathrm{ITL}_m = \bigcup_n \mathrm{ITL}_{m,n}$.

Next we show that agreement of words $u, v \in A^*$ on $\mathrm{ITL}_{m,n}$-formulae is the same as agreement on condensed rankers in $R_{m,n}$. We write $u \approx_{m,n} v$ if

$$u \models \varphi \;\Leftrightarrow\; v \models \varphi \quad \text{for all } \varphi \in \mathrm{ITL}_{m,n}.$$

For every fixed alphabet, the set $\mathrm{ITL}_{m,n}$ is finite up to equivalence. Thus $\approx_{m,n}$ is a finite index congruence.

Proposition 5. *Let $m, n \in \mathbb{N}$ and $u, v \in A^*$. Then $u \approx_{m,n} v$ if and only if $u \equiv_{m,n} v$.*

Corollary 3. *Let $L \subseteq A^*$ and $m \geq 2$. Then L is definable in ITL_m if and only if L is recognized by a monoid in $\mathbf{R}_m \vee \mathbf{L}_m$. In particular, it is decidable whether a given regular language is definable in ITL_m.*

Acknowledgments. We would like to thank the anonymous referees for several suggestions which helped to improve the presentation of the paper.

References

1. Almeida, J.: Finite Semigroups and Universal Algebra. World Scientific (1994)
2. Almeida, J., Azevedo, A.: The join of the pseudovarieties of $\mathcal{R}$-trivial and $\mathcal{L}$-trivial monoids. J. Pure Appl. Algebra 60, 129–137 (1989)
3. Birjukov, P.A.: Varieties of idempotent semigroups. Algebra Logika 9, 255–273 (1970)
4. Diekert, V., Gastin, P., Kufleitner, M.: A survey on small fragments of first-order logic over finite words. Int. J. Found. Comput. Sci. 19(3), 513–548 (2008)
5. Eilenberg, S.: Automata, Languages, and Machines, vol. B. Academic Press (1976)
6. Fennemore, C.F.: All varieties of bands I, II. Math. Nachr. 48, 237–262 (1971)
7. Gerhard, J.A.: The lattice of equational classes of idempotent semigroups. J. Algebra 15, 195–224 (1970)
8. Gerhard, J.A., Petrich, M.: Varieties of bands revisited. Proc. Lond. Math. Soc. 58(3), 323–350 (1989)
9. Krohn, K., Rhodes, J.L., Tilson, B.: Homomorphisms and semilocal theory. In: Algebraic Theory of Machines, Languages, and Semigroups, ch. 8, pp. 191–231. Academic Press (1968)
10. Kufleitner, M., Weil, P.: On the lattice of sub-pseudovarieties of **DA**. Semigroup Forum 81, 243–254 (2010)
11. Kufleitner, M., Weil, P.: The FO^2 alternation hierarchy is decidable. arXiv, abs/1203.6152 (2012)
12. Kufleitner, M., Weil, P.: On logical hierarchies within FO^2-definable languages. Log. Methods Comput. Sci. (in press, 2012)
13. Lodaya, K., Pandya, P.K., Shah, S.S.: Marking the chops: an unambiguous temporal logic. In: Proceedings of IFIP TCS 2008. IFIP, vol. 273, pp. 461–476. Springer (2008)
14. Pin, J.-É.: Propriétés Syntactiques du Produit Non Ambigu. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 483–499. Springer, Heidelberg (1980)
15. Pin, J.-É.: Varieties of Formal Languages. North Oxford Academic (1986)
16. Schützenberger, M.P.: Sur le produit de concaténation non ambigu. Semigroup Forum 13, 47–75 (1976)
17. Schwentick, T., Thérien, D., Vollmer, H.: Partially-Ordered Two-Way Automata: A New Characterization of DA. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 239–250. Springer, Heidelberg (2002)
18. Straubing, H.: Algebraic characterization of the alternation hierarchy in $\mathrm{FO}^2[<]$ on finite words. In: Proceedings of CSL 2011. LIPIcs, vol. 12, pp. 525–537. Dagstuhl Publishing (2011)
19. Tesson, P., Thérien, D.: Diamonds are forever: The variety DA. In: Proceedings of Semigroups, Algorithms, Automata and Languages 2001, pp. 475–500. World Scientific (2002)
20. Thérien, D., Wilke, Th.: Over words, two variables are as powerful as one quantifier alternation. In: Proceedings of STOC 1998, pp. 234–240. ACM Press (1998)
21. Trotter, P., Weil, P.: The lattice of pseudovarieties of idempotent semigroups and a non-regular analogue. Algebra Univers. 37(4), 491–526 (1997)
22. Weis, Ph., Immerman, N.: Structure theorem and strict alternation hierarchy for FO^2 on words. Log. Methods Comput. Sci. 5, 1–23 (2009)

Equations $X + A = B$ and $(X + X) + C = (X - X) + D$ over Sets of Natural Numbers

Tommi Lehtinen[1,2]

[1] Turku Centre for Computer Science
[2] Department of Mathematics, University of Turku, Finland
tommi.lehtinen@utu.fi

Abstract. It has recently been shown, that hyper-arithmetical sets can be represented as the unique solutions of language equations over sets of natural numbers with operations of addition, subtraction and union. It is shown that the same expressive power, under a certain encoding, can be achieved by systems of just two equations, $X + A = B$ and $(X + X) + C = (X - X) + D$, without using union. It follows that the problems concerning the solutions of systems of the general form are as hard as the same problems restricted to these systems with two equations, it is known that the question for solution existence is Σ_1^1 complete.

1 Introduction

Language equations are equations with formal languages as unknowns. They have lately been under active research, a survey on the topic has been written by Kunc [8]. This paper is focused on equations over unary alphabet, or, equivalently, over natural numbers.

Systems of language equations can be used to characterize formal languages. For example context-free languages can be represented as components of the least solutions of certain type of systems of equations. These were introduced by Ginsburg and Rice [1] and consists of equations

$$\begin{aligned} X_1 &= \phi_1(X_1, \ldots, X_n) \\ &\vdots \\ X_1 &= \phi_n(X_1, \ldots, X_n), \end{aligned}$$

where X_i are variables on formal languages over some fixed finite alphabet Σ, and ϕ_i are expressions with operations of union and concatenation of languages and finite constants. This kind of systems of equations, that have only an occurence of a variable on the left-hand side, are called *resolved*. If intersection is added to the set of operations the least solutions of such systems define the conjunctive languages introduced by Okhotin [12]. If also the negation is allowed, then the

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 615–629, 2012.

systems are able to represent the recursively enumerable languages as their least solutions, co-recursively enumerable languages as the greatest solutions, and recursive languages as the unique solutions [13].

If a more general form of systems of equations is considered

$$\begin{aligned}\phi_1(X_1,\ldots,X_n) &= \psi_1(X_1,\ldots,X_n)\\ &\vdots\\ \phi_m(X_1,\ldots,X_n) &= \psi_m(X_1,\ldots,X_n),\end{aligned}$$

called *unresolved* systems, where there can be any expressions on both sides of the equations, then the expressive power remains the same for systems where all Boolean operations are allowed. However, unresolved systems using more limited set of operations are a lot more powerful than resolved ones. Already systems with only the operations of union and concatenation can express all recursive (recursively enumerable, co-recursively enumerable) languages as their unique (least, greatest) solution [14], while their resolved counterpart can represent only the context-free languages.

The previous results concern languages over alphabets with multiple letters. In this setting non-commutativity is a fundamental property. The power of non-commutation was underlined by the result of Kunc [7], who showed that computational universality occurs in extremely simple equations $LX = XL$, only concerning commutation with a finite language $L \subset \{a,b\}^*$. The greatest solution of such an equation can be co-recursively enumerably universal.

In the case of a unary alphabet $\Sigma = \{a\}$, unary words a^k are in one to one correspondence with their lengths. Moreover, the length of concatenation of two words is the sum of their lengths. So unary words can be considered as natural numbers, and the algebraic structure remains the same. Sets of numbers take the place of unary languages, $\{k \in \mathbb{N} \mid a^k \in L\}$ instead of L. The notions of natural numbers and unary words can be used to represent the same thing. This paper uses natural numbers, as the notation is simpler, but everything could be stated also in terms of unary languages.

The concatenation of unary languages (or addition in terms of numbers) is commutative, so one cannot take advantage of non-commutativity when expressing complicated objects. For example the commutation equation $LX = XL$ is always trivially true in this case. Also the resolved systems with operations of concatenation and union can only define regular languages in the unary case.

Leiss [11] had constructed an equation over unary alphabet, with operations of concatenation and complementation, that has a non-regular solution. Still the expressive power of equations over a unary alphabet seemed quite limited until Jeż [2] proved that conjuctive languages over unary alphabet can be non-regular. Jeż and Okhotin [5] were able to use similar methods than Jeż [2] to prove that, in fact, unresolved systems of equations over sets of natural numbers with the operations of union and addition are computationally complete.

Jeż and Okhotin [3] further showed that computational universality can be achieved by systems of equations that use only the operation of addition and eventually periodic constants. These systems cannot represent all recursive (r.e., co-r.e.) languages [9], but they can represent *encodings* of these languages. Here an encoding S' of a set of numbers S means, that there is an embedding $\tau\colon \mathbb{N} \to \mathbb{N}$, such that $n \in S$ if and only if $\tau(n) \in S'$.

Lehtinen and Okhotin [10] finally showed that encodings of recursive (r.e., co-r.e.) can be represented as the unique (least, greatest) solutions of systems of equations using only one variable, the operation of addition and two equations

$$\begin{aligned} X + X + A &= X + X + B \\ X + C &= D, \end{aligned}$$

where A, B, C and D are eventually periodic constants. In this paper a similar result is presented concerning systems of equations with quotient.

In the context of natural numbers, the operation corresponding to quotient is subtraction, defined by $A - B = \{a - b \mid a \in A, b \in B, a \geqslant b\}$ for $A, B \subseteq \mathbb{N}$. Jeż and Okhotin [6] studied systems of equations over natural numbers with the operations of addition, subtraction and union. They found out, that the unique solutions of such systems can represent exactly the *hyper-arithmetical* sets.

Hyper-arithmetical sets Δ_1^1 are defined as the intersection $\Sigma_1^1 \cap \Pi_1^1$ of the two bottom classes of the analytical hierarchy.

In this paper it is shown, that these systems of equations over sets of natural numbers can be simulated by univariate systems of equations with the operations of addition and subtraction, and eventually periodic constants. The number of equations in the systems can be limited to two, the systems being of the form

$$\begin{aligned} X + A &= B \\ (X + X) + C &= (X - X) + D. \end{aligned}$$

Thus all hyper-arithmetical sets can be represented, although only in encoded form, by the unique solutions of this kind of simple systems of equations. Solutions of these systems are in one to one correspondence to the systems of more general form, and thus inherit the complexity of decision problem on the existence of solutions from the general case, which is Σ_1^1-complete.

The encodings used in [10] were able to encode only sets including zeroes. The encoding presented in this paper does not have the same limitation.

2 Simulating Many Variables by One

The main object in this paper is the set of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$. Focus is on systems of equations over the subsets of natural numbers $\mathcal{P}(\mathbb{N})$. Addition and subtraction of numbers can be defined to these sets of numbers by $A + B = \{a + b \mid a \in A, b \in B\}$ and $A - B = \{a - b \mid a \in A, b \in B, a \geqslant b\}$ for $A, B \subseteq \mathbb{N}$. These operations correspond to concatenation and quotient of languages, when a set of natural numbers is considered as a language over a unary alphabet. More precisely, if $\widehat{A} = \{a^k \mid k \in A\}$ and $\widehat{B} = \{a^k \mid k \in B\}$, then

$\widehat{A} \cdot \widehat{B} = \{a^k \mid k \in A + B\}$ and $\widehat{A} \cdot \widehat{B}^{-1} = \{a^k \mid k \in A - B\}$, so the structures of sets of natural numbers and unary languages are isomorphic.

Systems of equations over sets of natural numbers are sets of equations

$$\phi(X_1, \ldots, X_n) = \psi(X_1, \ldots, X_n),$$

where ϕ and ψ are expressions using the variables, and operations and constant sets from some fixed sets. An m-tuple $(S_1, \ldots, S_n)$ is a solution of the system, if $\phi(S_1, \ldots, S_n) = \psi(S_1, \ldots, S_n)$ holds for every equation in the system.

The goal of this section is to simulate a multivariable system using addition, subtraction and union by a system with one variable, and without using union. First the system of equations to be simulated should be fixed.

For $I \subseteq \{1, 2, \ldots, m-1\} \times \{1, 2, \ldots, m-1\}$, $J \subseteq \{1, 2, \ldots, m-1\}$ and eventually periodic $F \subseteq \mathbb{N}$, define the expressions

$$\phi_{(I,J,F)}(X_1, \ldots, X_{m-1}) = \bigcup_{(i_1,i_2)\in I} (X_{i_1} + X_{i_2}) \cup \bigcup_{j\in J} X_j \cup F$$

and

$$\psi_{(I,J,F)}(X_1, \ldots, X_{m-1}) = \bigcup_{(i_1,i_2)\in I} (X_{i_1} - X_{i_2}) \cup \bigcup_{j\in J} X_j \cup F.$$

These expressions will be the left and right-hand sides of the equations in the system.

So the starting point is a system of equations of $m-1$ variables, with equations of the form

$$\phi_{(I_1,J_1,F_1)}(X_1, \ldots, X_{m-1}) = \psi_{(I_2,J_2,F_2)}(X_1, \ldots, X_{m-1})$$

where $I_1, I_2 \subseteq \{1, 2, \ldots, m-1\} \times \{1, 2, \ldots, m-1\}$, $J_1, J_2 \subseteq \{1, 2, \ldots, m-1\}$, $F_1, F_2 \subseteq \mathbb{N}$ are eventually periodic constant sets. The index set $\mathcal{I}$, that contains pairs of elements from

$$\mathcal{P}(\{1, \ldots, m-1\} \times \{1, \ldots, m-1\}) \times \mathcal{P}(\{1, \ldots, m-1\}) \times \mathcal{F},$$

where $\mathcal{F}$ is the set of eventually periodic subsets of natural numbers, is used to specify the system.

It is easy to see by possibly introducing new variables, that any system using the operations of addition, subtraction and union can be transformed into this kind of an equation in such a way, that the solutions stay the same. This system will be simulated by a system with only one variable, using addition and subtraction as the only operations.

The encoding uses the structure of the natural numbers. If p is a positive number, then define the embeddings $\tau_i^p : \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ as $\tau_i^p(S) = \{pn + i \mid n \in S\}$.

For any set S the numbers of the form $pn + i$ are referred to as the *track* i of S. Track i is said to contain the set $\{n \in \mathbb{N} \mid pn + i \in S\}$.

The encoding $\pi : \mathcal{P}(\mathbb{N})^{m-1} \to \mathcal{P}(\mathbb{N})$ for the simulation is defined by

$$\begin{aligned} \pi(S_1, \dots, S_{m-1}) = {} & (\bigcup_{i=0}^{2m} \tau_i^{\mathfrak{p}}(\mathbb{N}))) \cup \\ & (\bigcup_{k=1}^{m-1} \tau_{(3m+k)}^{\mathfrak{p}}(S_k)) \cup \\ & (\bigcup_{k=1}^{m-1} \tau_{(k\mathfrak{b})}^{\mathfrak{p}}(S_k)) \cup \\ & \tau_{4m}^{\mathfrak{p}}(\{0\}) \cup \tau_{\mathfrak{b}m}^{\mathfrak{p}}(\{0\}) \end{aligned}$$

where $\mathfrak{b} = 4(2m+1)$, $\mathfrak{p} = (m+1)\mathfrak{b}$ and $\mathfrak{c} = p - 4m - 1 = \mathfrak{b}m + 4m + 3$.

Suppose $S = \pi(S_1, \dots, S_{m-1})$ for some $S_1, \dots, S_{m-1} \subseteq \mathbb{N}$. Then the set S_k is encoded on tracks $3m + k$ and $\mathfrak{b}k$. To simulate the system of equations with multible variables, the sums and differences of two sets S_k and S_ℓ are needed. They are encoded on tracks $\mathfrak{b}k + 3m + \ell$ of $S + S$ and $\mathfrak{b}k - 3m - \ell$ of $S - S$, respectively.

To compare the contents of individual tracks, useless information must be overwritten. For that purpose, define the three sets

$$\begin{aligned} E &= (\bigcup_{i=0}^{4m} \{(2m+1)i\}) \cup \{\mathfrak{b}m + 2, \mathfrak{b}m + 2m + 2\} \cup \{\mathfrak{c} + 1, \mathfrak{c} + 2m + 1\} \\ E^+ &= \bigcup_{i=0}^{m-1} \{\mathfrak{b}i, \mathfrak{b}i + 4m + 4\} \cup \{\mathfrak{b}m - 2m + 2, \mathfrak{b}m + 4m + 4\} \\ E^- &= \bigcup_{i=0}^{m-1} (\{\mathfrak{b}i\} + \{0, 2, 2m+2, 6m+4\}). \end{aligned}$$

Adding these to S, $S + S$ and $S - S$ results in a set that has all other tracks full, except track $\mathfrak{c}$, which remains empty. This is stated in the following lemma. The proof is omitted due to space constraints.

Lemma 1. *If* $S = \pi(S_1, S_2, \dots, S_{m-1})$ *for some* $S_i \subseteq \mathbb{N}$*, then* $S + E = S + S + E^+ = (S - S) + E^- = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N})$ □

Now the track $\mathfrak{c}$ can be used to represent expressions over over $S_1, S_2, \dots, S_{m-1}$. For example, the track $\mathfrak{b}k + 3m + \ell$ of $S + S$ contains the sum $S_k + S_\ell$. Adding this to $\mathfrak{c} - (\mathfrak{b}k + 3m + \ell)$ moves it to track $\mathfrak{c}$, so that

$$S + S + (E^+ \cup \{\mathfrak{c} - (\mathfrak{b}k + 3m + \ell)\}) = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(S_k + S_\ell).$$

This makes it possible to express the expressions used in the system being simulated. For each expression

$$\phi_{(I,J,F)}(X_1, \dots, X_{m-1}) = \bigcup_{(i_1, i_2) \in I} (X_{i_1} + X_{i_2}) \cup \bigcup_{j \in J} X_j \cup F$$

appearing in the system, define a set

$$E^+_{(I,J,F)} = E^+ \cup \{\mathfrak{c} - (\mathfrak{b}i_1 + 3m + i_2) \mid (i_1, i_2) \in I\} \cup \{\mathfrak{c} - (\mathfrak{b}m + 3m + j) \mid j \in J\} \cup \tau^{\mathfrak{p}}_{(\mathfrak{c}-\mathfrak{b}m-4m)}(F).$$

Similarly, for each expression

$$\psi_{(I,J,F)}(X_1, \ldots, X_{m-1}) = \bigcup_{(i_1,i_2)\in I} (X_{i_1} - X_{i_2}) \cup \bigcup_{j \in J} X_j \cup F,$$

define the set

$$E^-_{(I,J,F)} = E^- \cup \{\mathfrak{c} - (\mathfrak{b}i_1 - (3m + i_2)) \mid (i_1, i_2) \in I\} \cup \{\mathfrak{c} - (\mathfrak{b}j - 4m) \mid j \in J\} \cup \tau^{\mathfrak{p}}_{(\mathfrak{c}-\mathfrak{b}m+4m)}(F).$$

When these sets are added to $S + S$ and $S - S$, all other tracks will be full, except track $\mathfrak{c}$ will contain the encoding of $\phi_{(I,J,F)}(S_1, \ldots, S_{m-1})$ or $\psi_{(I,J,F)}(S_1, \ldots, S_{m-1})$. This is stated in the next lemma.

Lemma 2. *Let $S = \pi(S_1, S_2, \ldots, S_{m-1})$ for some $S_i \subseteq \mathbb{N}$. Then*

$$S + S + E^+_{(I,J,F)} = \bigcup_{i \neq \mathfrak{c}} \tau^{\mathfrak{p}}_i(\mathbb{N}) \cup \tau^{\mathfrak{p}}_{\mathfrak{c}}(\phi_{(I,J,F)}(S_1, \ldots, S_{m-1}))$$

and

$$(S - S) + E^-_{(I,J,F)} = \bigcup_{i \neq \mathfrak{c}} \tau^{\mathfrak{p}}_i(\mathbb{N}) \cup \tau^{\mathfrak{p}}_{\mathfrak{c}}(\psi_{(I,J,F)}(S_1, \ldots, S_{m-1})).$$

Proof. First consider $S + S + E^+_{(I,J,F)}$.

By Lemma 1, $S + S + E^+ = \bigcup_{i \neq \mathfrak{c}} \tau^{\mathfrak{p}}_i(\mathbb{N})$.

Track $\mathfrak{b}i_1 + 3m + i_2$ of $S + S$ contains the encoding of $S_{i_1} + S_{i_2}$, track $\mathfrak{b}m + 3m + j$ contains S_j and track $\mathfrak{b}m + 4m$ contains $\{0\}$. The proof that these are the exact contents on these tracks will be omitted here.

Adding these to $\mathfrak{c} - (\mathfrak{b}i_1 - (3m + i_2))$ takes $S_{i_1} + S_{i_2}$ on track $\mathfrak{c}$. similarly, adding to $\mathfrak{c} - (\mathfrak{b}j - 4m)$ takes S_j on track $\mathfrak{c}$, and adding $\tau^{\mathfrak{p}}_{(\mathfrak{c}-\mathfrak{b}m+4m)}(F)$ takes F to the same track.

The union of these is $\phi_{(I,J,F)}(S_1, \ldots, S_{m-1})$, and hence

$$S + S + E^+_{(I,J,F)} = \bigcup_{i \neq \mathfrak{c}} \tau^{\mathfrak{p}}_i(\mathbb{N}) \cup \tau^{\mathfrak{p}}_{\mathfrak{c}}(\phi_{(I,J,F)}(S_1, \ldots, S_{m-1})).$$

The other case of

$$(S - S) + E^-_{(I,J,F)} = \bigcup_{i \neq \mathfrak{c}} \tau^{\mathfrak{p}}_i(\mathbb{N}) \cup \tau^{\mathfrak{p}}_{\mathfrak{c}}(\psi_{(I,J,F)}(S_1, \ldots, S_{m-1})).$$

is proved in a similar manner.

The fact that the encoded expressions are on the same track, all other tracks being full, makes it easy to compare the expessions:

Lemma 3. *Let $S = \pi(S_1, \dots, S_{m-1})$ for some $S_i \subseteq \mathbb{N}$. Then the equation*

$$\phi_{(I_1,J_1,F_1)}(X_1, \dots, X_{m-1}) = \psi_{(I_2,J_2,F_2)}(X_1, \dots, X_{m-1})$$

has $(S_1, \dots, S_{m-1})$ as a solution if and only if the equation

$$X + X + E^{+}_{(I_1,J_1,F_1)} = (X - X) + E^{-}_{(I_2,J_2,F_2)}$$

has S as a solution.

Proof. By Lemma 2

$$S + S + E^{+}_{(I_1,J_1,F_1)} = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(\phi_{(I_1,J_1,F_1)}(S_1, \dots, S_{m-1}))$$

and

$$(S - S) + E^{-}_{(I_2,J_2,F_2)} = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(\psi_{(I_2,J_2,F_2)}(S_1, \dots, S_{m-1})).$$

It follows that

$$S + S + E^{+}_{(I_1,J_1,F_1)} = (S - S) + E^{-}_{(I_2,J_2,F_2)}$$

if and only if

$$\tau_{\mathfrak{c}}^{\mathfrak{p}}(\phi_{(I_1,J_1,F_1)}(S_1, \dots, S_{m-1})) = \tau_{\mathfrak{c}}^{\mathfrak{p}}(\psi_{(I_2,J_2,F_2)}(S_1, \dots, S_{m-1})).$$

This is equivalent to

$$\phi_{(I_1,J_1,F_1)}(S_1, \dots, S_{m-1}) = \psi_{(I_2,J_2,F_2)}(S_1, \dots, S_{m-1}),$$

which was claimed. □

The correspondence of the equations in the two systems requires S to be an encoding of some sets. To make the simulating system complete, there needs to be equations guaranteeing this indeed is the case.

The following lemma states a set of equations that has $S \subseteq \mathbb{N}$ as a solution only if it is almost a correct encoding of some sets. The only thing missing is, that the two sets of data tracks do not have to have equal content. The set

$$E_{\varnothing} = E \cup \big(\{\mathfrak{c}\} - (\{0, \dots, \mathfrak{c}\} \setminus (\{0, \dots, 2m\} \cup (\bigcup_{k=1}^{m} \{(3m+k), k\mathfrak{b}\}) \cup \{\mathfrak{c}+1, \dots, \mathfrak{p}-1\}\big),$$

is used to declare one of the equations.

Lemma 4. *If a set of natural numbers S is a solution of the equations*

$$X + E_{\varnothing} = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \tag{1}$$

$$X + (E \cup \{\mathfrak{c} - i\}) = \mathbb{N}, \textit{ for each } i = 0, \dots, 2m \tag{2}$$

$$X + (E \cup \{\mathfrak{c} - 4m\}) = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(\{0\}) \tag{3}$$

$$X + (E \cup \{\mathfrak{c} - \mathfrak{b}m\}) = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(\{0\}), \tag{4}$$

then $\pi(\varnothing, \dots, \varnothing) \subseteq S \subseteq \pi(\mathbb{N}, \dots, \mathbb{N})$.

The proof is omitted here. Another set of equations is needed to check, that the tracks $3m+k$ and $\mathfrak{b}k$ of S have the same content. The next lemma states these equations, the proof is omitted again.

Lemma 5. *Let S satisfy the equations in Lemma 4 and the equation*

$$S + S + (E^+ \cup \{\mathfrak{c} - (\mathfrak{b}m + 3m + k)\}) = (S - S) + (E^- \cup \{\mathfrak{c} - (\mathfrak{b}k - 4m)\})$$

Then the tracks $3m+k$ and $\mathfrak{b}k$ of S have the same content. If this holds for all $k = 1, 2, \ldots, m$, then $S = \pi(S_1, \ldots, S_{m-1})$ for some sets $S_i \subseteq \mathbb{N}$. □

The above Lemmas 4 and 5 state equations that have valid encodings of $m-1$ sets as their solutions. Lemma 3 states equations that have such encodings as solutions if and only if the equations of the original system has the encoded sets as solutions. The next theorem sums up the correctness of the simulation.

Theorem 1. *A set $S \subseteq \mathbb{N}$ is a solution of the equations*

$$X + E_{\varnothing} = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N})$$

$$X + (E \cup \{\mathfrak{c} - i\}) = \mathbb{N}, \textit{ for each } i = 0, \ldots, 2m$$

$$X + (E \cup \{\mathfrak{c} - 4m\}) = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(\{0\})$$

$$X + (E \cup \{\mathfrak{c} - \mathfrak{b}m\}) = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(\{0\})$$

$$X + X + (E^+ \cup \{\mathfrak{c} - (\mathfrak{b}m + 3m + k)\}) = (X - X) + (E^- \cup \{\mathfrak{c} - (\mathfrak{b}k - 4m)\}), \textit{ for } k = 1, \ldots, m-1,$$

and

$$X + X + E^+_{(I_1,J_1,F_1)} = (X - X) + E^-_{(I_2,J_2,F_2)}, \textit{ for } ((I_1,J_1,F_1),(I_2,J_2,F_2)) \in \mathcal{I}$$

if and only if $S = \pi(S_1, \ldots, s_{m-1})$ for some $S_i \subseteq \mathbb{N}$ such that $(S_1, \ldots, S_{m-1})$ is a solution of the equations

$$\phi_{(I_1,J_1,F_1)}(X_1, \ldots, X_m) = \psi_{(I_2,J_2,F_2)}(X_1, \ldots, X_m), \textit{ for } ((I_1,J_1,F_1),(I_2,J_2,F_2)) \in \mathcal{I}.$$

□

The solutions of these two systems are in one to one correspondence, so it essentially irrelevant which of them is solved. Moreover, the encoding $\pi(S_1, \ldots, S_{m-1}) \subseteq \pi(S'_1, \ldots, S'_{m-1})$ if and only if $S_i \subseteq S'_i$ for all i. In other words, the order of solutions is preserved in the simulation as well.

3 Two Equations

In the previous section a systems of equations were constructed. These systems consist of equations of unified forms of two types.

Suppose

$$X + A_i = B_i, \text{ for } i = 1, 2, \ldots, e_1$$
$$X + X + C_i = (X - X) + D_i, \text{ for } i = 1, 2, \ldots, e_2,$$

where A_i, B_i, C_i and D_i are eventually periodic constants, is an arbitrary fixed system of that kind. This system will be simulated by a system with only two equations.

Let $\mathfrak{p} = \max(e_1, e_2) + 1$.

In this system all the equations of the same type are handled by one equation. This can be achieved by taking the constants of the above system, and defining new constants by letting the old ones be on different tracks. That is, the new constants are defined by:

$$A = \bigcup_{i=1}^{e_1} \tau_i^{\mathfrak{p}}(A_i)$$
$$B = \bigcup_{i=1}^{e_1} \tau_i^{\mathfrak{p}}(B_i)$$
$$C = \bigcup_{i=1}^{e_2} \tau_i^{\mathfrak{p}}(C_i)$$
$$D = \bigcup_{i=1}^{e_2} \tau_i^{\mathfrak{p}}(D_i)$$

Lemma 6. *The equation $\tau_0^{\mathfrak{p}}(S) + A = B$ holds if and only if the equations $S + A_i = B_i$ for $i = 1, 2, \ldots, e_1$ hold.* □

Proof. By the definition of A $\tau_0^{\mathfrak{p}}(S) + A = \bigcup_{i=1}^{e_1} \tau_i^{\mathfrak{p}}(S + A_i)$. This is the same as $B = \bigcup_{i=1}^{e_1} \tau_i^{\mathfrak{p}}(B_i)$ if and only if they are equal on all tracks $i = 1, \ldots, e_1$, that is if and only if all equations $S + A_i = B_i$ hold.

Lemma 7. *The equation $\tau_0^{\mathfrak{p}}(S) + \tau_0^{\mathfrak{p}}(S) + C = (\tau_0^{\mathfrak{p}}(S) - \tau_0^{\mathfrak{p}}(S)) + D$ holds if and only if the equations $S + S + C_i = (S - S) + D_i$ for $i = 1, 2, \ldots, e_2$ hold.*

Proof. As in the proof of the previous lemma,

$$\tau_0^{\mathfrak{p}}(S) + \tau_0^{\mathfrak{p}}(S) + C = \bigcup_{i=1}^{e_2} \tau_i^{\mathfrak{p}}(S + S + C_i)$$

and

$$(\tau_0^{\mathfrak{p}}(S) - \tau_0^{\mathfrak{p}}(S)) + D = \bigcup_{i=1}^{e_2} \tau_i^{\mathfrak{p}}((S - S) + D_i).$$

The sets are equal if and only if the equations $S + S + C_i = (S - S) + D_i$ for $i = 1, 2, \ldots, e_2$ hold, which was claimed. □

Lemma 8. *A set $S \subseteq \mathbb{N}$ is the solution of*

$$X + A = B$$
$$X + X + C = (X - X) + D$$

if and only if $S = \tau_0^{\mathfrak{p}}(S')$ for some solution S' of

$$X + A_i = B_i, \textit{ for } i = 1, 2, \ldots, e_1$$
$$X + X + C_i = (X - X) + D_i, \textit{ for } i = 1, 2, \ldots, e_2.$$

Proof. Let S' be a solution of

$$X + A_i = B_i, \text{ for } i = 1, 2, \ldots, e_1$$
$$X + X + C_i = (X - X) + D_i, \text{ for } i = 1, 2, \ldots, e_2,$$

and $S = \tau_0^{\mathfrak{p}}(S')$. Then S' is a solution of the equations $X_k + A_i = B_i$ and by Lemma 6 S is a solution of $X + A = B$. At the same time S' is a solution of of all $X + X + C_i = (X - X) + D_i$, and thus, by Lemma 7, $X + X + C = (X - X) + D$ has S as a solution as well.

Conversely, assume that $S + A = B$ and $S + S + C = (S - S) + D$. Let $\mathfrak{p}n + a \in S$, where $0 \leqslant a < \mathfrak{p}$. If $\mathfrak{p} - m \leqslant a$, then $\mathfrak{p}n' - a \in A$ for some $n' \in \mathbb{N}$ and $(\mathfrak{p}n + a) + (\mathfrak{p}n' - a) = \mathfrak{p}(n + n') \in B$. This is a contradiction, since track 0 of B is empty.

If $0 < a < \mathfrak{p} - m$, then $(\mathfrak{p}n + a) + (\mathfrak{p}n' + m) = \mathfrak{p}(n + n') + (m + a) \in B$, as $\mathfrak{p}n' + m \in A$ for some $n' \in \mathbb{N}$ by the definition of A. This is again a contradiction, because $m < m + a < \mathfrak{p}$ and the corresponding tracks are empty in B.

So it must be the case that $a = 0$, and thus $S = \tau_0^{\mathfrak{p}}(S')$ for some $S' \subseteq \mathbb{N}$. It follows from Lemmas 6 and 7, that S' is a solution of

$$X + A_i = B_i, \text{ for } i = 1, 2, \ldots, e_1$$
$$X + X + C_i = (X - X) + D_i, \text{ for } i = 1, 2, \ldots, e_2,$$

which completes the proof the lemma. □

Theorem 1 and the construction in this section together yield the following theorem.

Theorem 2. *Let*

$$\phi_i(X_1, \ldots, X_m) = \psi_i(X_1, \ldots, X_m), \textit{ for } i \in I.$$

be a system of equations.

There exist such eventually periodic sets A, B, C and D, and numbers $0 < d_i < p$, that $(S_1, \ldots, s_{m-1})$ is a solution of the above equation if and only if there is a solution S of

$$X + A = B$$
$$X + X + C = (X - X) + D,$$

such that $S \cap \tau_{d_i}^p(\mathbb{N}) = \tau_{d_i}^p(S_{d_i})$, for $i \in I$. □

The correspondence of the solutions is one to one, and preserves order. Hence a system has a unique solution if and only if the constructed system has a unique solution. Also the least and greatest solution exist for these equations at the same time.

4 Conclusion

It has been shown that systems of two equations

$$X + A = B$$
$$X + X + C = (X - X) + D$$

over the sets of natural numbers can simulate any systems of equations over sets of natural numbers using the operations of addition, subtraction and union. The solutions of any simulated system over multiple variables are in one to one correspondence with the solutions of the constructed simple system.

Consequently, solving questions about system of two equations using only addition and subtraction are as hard as solving the same questions in the general case. The unique solutions of the general systems can represent hyperarithmetical sets and the problem for solution existence is Σ^1_1-complete. The systems of the stated simple form can represent the same sets in an encoded form, that is n is in some set S if and only if $n \cdot p + i$, for some fixed p and i, is in the corresponding set containing the encoding of S, and solution existence is, as mentioned above, the same, Σ^1_1-complete.

The high expressive power of these simple systems seem to imply, that no practical (in the sense of computational complexity) families of languages can be defined by this kind of systems. Allthough, restricting the set of constants from all regular sets to some subfamily might still lead to a some practically computable subclass.

Also the expressive power of the general systems by the least and greatest solutions is an open question, while it is expected these are the Σ^1_1- and Π^1_1-sets. Whatever they turn out to be, they can be represented in an encoded form by the simple system of two equations presented in this paper.

References

1. Ginsburg, S., Rice, H.G.: Two families of languages related to ALGOL. Journal of the ACM 9, 350–371 (1962)
2. Jeż, A.: Conjunctive grammars can generate non-regular unary languages. International Journal of Foundations of Computer Science 19(3), 597–615 (2008)
3. Jeż, A., Okhotin, A.: Equations over sets of natural numbers with addition only. In: STACS 2009, Freiburg, Germany, February 26-28, pp. 577–588 (2009)
4. Jeż, A., Okhotin, A.: Least and Greatest Solutions of Equations over Sets of Integers. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 441–452. Springer, Heidelberg (2010)

5. Jeż, A., Okhotin, A.: On the Computational Completeness of Equations over Sets of Natural Numbers. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 63–74. Springer, Heidelberg (2008)
6. Jeż, A., Okhotin, A.: Representing hyper-arithmetical sets by equations over sets of integers. Theory of Computing Systems (accepted)
7. Kunc, M.: The power of commuting with finite sets of words. Theory of Computing Systems 40(4), 521–551 (2007)
8. Kunc, M.: What Do We Know About Language Equations? In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 23–27. Springer, Heidelberg (2007)
9. Lehtinen, T., Okhotin, A.: On equations over sets of numbers and their limitations. International Journal of Foundations of Computer Science 22(2), 377–393 (2011)
10. Lehtinen, T., Okhotin, A.: On Language Equations $XXK = XXL$ and $XM = N$ over a Unary Alphabet. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 291–302. Springer, Heidelberg (2010)
11. Leiss, E.L.: Unrestricted complementation in language equations over a one-letter alphabet. Theoretical Computer Science 132, 71–93 (1994)
12. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
13. Okhotin, A.: Decision problems for language equations. Journal of Computer and System Sciences 76(3-4), 251–266 (2010)
14. Okhotin, A.: Unresolved systems of language equations: expressive power and decision problems. Theoretical Computer Science 349(3), 283–308 (2005)

Appendix

A Proofs of Lemmas 1, 4 and 5

Lemma 1. If $S = \pi(S_1, S_2, \ldots, S_{m-1})$ for some $S_i \subseteq \mathbb{N}$, then $S + E = S + S + E_+ = (S - S) + E_- = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N})$

Proof. Case of $S + E$:

The tracks $0, 1, \ldots, 2m$ of S are full and sums with $\bigcup_{i=0}^{4m}\{(2m+1)i\}$ give full tracks $0, 1, \ldots, \mathfrak{b}m + 2m$. Then sums with $\{\mathfrak{b}m + 2, \mathfrak{b}m + 2m + 2\}$ give full tracks $\mathfrak{b}m + 2, \mathfrak{b}m + 3, \ldots, \mathfrak{c} - 1$ and sums with $\{\mathfrak{c} + 1, \mathfrak{b}m + 6m + 4\}$ give tracks $\mathfrak{c} + 1, \mathfrak{c} + 2, \ldots, \mathfrak{p} - 1$.

Track $\mathfrak{c}$ remains empty by the first equation of Lemma 4.

Case of $S + S + E_+$:

The tracks $0, 1, \ldots, 6m$ of $S + S$ are full. The sum with $\bigcup_{i=0}^{m-1}\{\mathfrak{b}i, \mathfrak{b}i + 4m + 4\}$ gives full tracks $0, 1, \ldots, \mathfrak{b}(m-1) + 4m + 4 + 6m$, that is $0, 1, \ldots, \mathfrak{b}m + 2m$. The sum with $\mathfrak{b}m - 2m + 2$ makes tracks $\mathfrak{b}m - 2m + 2, \ldots, \mathfrak{b}m + 4m + 2 (= \mathfrak{c} - 1)$ full, and the sum with $\mathfrak{b}m + 4m + 4 = \mathfrak{c} + 1$ makes tracks $\mathfrak{c} + 1, \mathfrak{c} + 2, \ldots, \mathfrak{p} - 1$ full.

The track $\mathfrak{c}$ remains empty, as the tracks $\mathfrak{c} - \mathfrak{b}i = \mathfrak{b}(m - i) + 4m + 3$ and $\mathfrak{c} - (\mathfrak{b}i - 4m - 4) = \mathfrak{b}(m - i) - 1$ of $S + S$ are empty for $i = 0, 1, \ldots, m - 1$, as well as tracks $\mathfrak{c} - (\mathfrak{b}m - 2m + 2) = 6m + 1$ and $\mathfrak{c} - (\mathfrak{b}m + 4m + 4) = -1$, which is the track $\mathfrak{p} - 1$.

Case of $(S - S) + E_-$:

In the set $S - S$ the tracks $0, \ldots, 2m$ are full. Summing these with E_- gives full tracks $\bigcup_{i=0}^{m-1}\big(\mathfrak{b}i + (\{0, 1, \ldots, 4m + 2\} \cup \{6m + 4, 6m + 5, \ldots 8m + 4\})\big)$. Since $\mathfrak{b}m \in S$, also the tracks $(\mathfrak{p} + \{0, \ldots, 2m\}) - \mathfrak{b}m = \mathfrak{b} + \{0, \ldots, 2m\}$ are full in $S - S$. The sum of these with $\mathfrak{b}(m - 1) + \{0, 2, 2m + 2, 6m + 4\} \in E_-$ give full tracks $\mathfrak{b}m + (\{0, 1, \ldots, 4m + 2\} \cup \{6m + 4, 6m + 5, \ldots 8m + 4\})$. So all tracks but $\mathfrak{b}i + \{4m + 3, 4m + 4, \ldots, 6m + 3\}$ for $i = 0, 1, \ldots, m$ are known to be full in $(S - S) + E_-$.

The tracks $\mathfrak{p} - 4m, \mathfrak{p} - 4m + 1, \ldots, \mathfrak{p} - 1$ are full in $S - S$, since $0, 2m, 4m \in S$ and the tracks $0, 1, \ldots 2m$ are full in S. This yields full tracks $\mathfrak{b}m + 4m + 4 (= \mathfrak{p} - 4m), \mathfrak{b}m + 4m + 5, \ldots, \mathfrak{b}m + 6m + 3$. Furthermore, the sums with $\mathfrak{b}i + 6m + 4 \in E_-$ makes tracks $\mathfrak{b}i + 4m + 3, \mathfrak{b}i + 4m + 4, \ldots, \mathfrak{b}i + 6m + 3$ almost full, possibly missing only the first number. The tracks $2m + 1, 2m + 2, \ldots, 4m$ of $S - S$ contain at least 0 and sums with $\mathfrak{b}i + 2m + 2 \in E_-$ give $\mathfrak{b}i + \{4m + 3, 4m + 4, \ldots, 6m + 3\} \in (S - S) + E_-$ for $i = 0, 1, \ldots, m - 1$.

So all tracks except $\mathfrak{c} = \mathfrak{b}m + 4m + 3$ are full. For this track to be empty in $(S - S) + E_-$, the tracks $\mathfrak{b}m + 4m + 3 - (\{\mathfrak{b}i\} + \{0, 2, 2m + 2, 6m + 4\})$ need to be empty in $S - S$ for $i = 0, 1, \ldots, m - 1$. In different notation these are the tracks $\mathfrak{b}j + \{-2m - 1, 2m + 1, 4m + 1, 4m + 3\}$ for $j = 1, 2, \ldots, m$, which indeed are empty in $S - S$. □

Lemma 4. If a set of natural numbers S is a solution of the equations

$$X + E_{\varnothing} = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \tag{5}$$

$$X + (E \cup \{\mathfrak{c} - i\}) = \mathbb{N}, \text{ for each } i = 0, \dots, 2m \tag{6}$$

$$X + (E \cup \{\mathfrak{c} - 4m\}) = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(\{0\}) \tag{7}$$

$$X + (E \cup \{\mathfrak{c} - \mathfrak{b}m\}) = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(\{0\}), \tag{8}$$

then $\pi(\varnothing, \dots, \varnothing) \subseteq S \subseteq \pi(\mathbb{N}, \dots, \mathbb{N})$.

Proof. Suppose S is a solution of the equations.

The first equation implies that the tracks of S with numbers in $\{0, 1, \dots, \mathfrak{c}\} \setminus (\{0, 1, \dots, 2m\} \cup (\bigcup_{k=1}^{m} \{(3m + k), k\mathfrak{b}\}))) \cup \{\mathfrak{c} + 1, \mathfrak{c} + 2, \dots, \mathfrak{p} - 1\}$ are empty. If one of them, say track j, would contain a number, this number added to $\mathfrak{c} - j$ would end up on track $\mathfrak{c}$, which is empty.

Similarly the tracks $\mathfrak{c} + 1, \mathfrak{c} + 2, \dots, \mathfrak{p} - 1$ of S are empty, as otherwise the track $\mathfrak{c}$ would not be empty in the sum. This follows as

$$(\mathfrak{c} + 1) + (\mathfrak{p} - 1) = (\mathfrak{c} + 2) + (\mathfrak{p} - 2) = \dots = (\mathfrak{p} - 1) + (\mathfrak{c} + 1) = \mathfrak{c} + \mathfrak{p}$$

and $\mathfrak{c} + \mathfrak{p}$ is the same track as $\mathfrak{c}$.

Since $S + E \subseteq \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N})$ by the first equation, the equations $S + (E \cup \{\mathfrak{c} - i\}) = \mathbb{N}$ quarantee that the tracks $0, 1, \dots, 2m$ are full.

Similarly the last two equations ensure that the tracks $4m$ and $\mathfrak{b}m$ contain the encoding of $\{0\}$.

It follows that $\pi(\varnothing, \dots, \varnothing) \subseteq S \subseteq \pi(\mathbb{N}, \dots, \mathbb{N})$. □

Lemma 5. Let S satisfy the equations in Lemma 4 and the equation

$$S + S + (E^{+} \cup \{\mathfrak{c} - (\mathfrak{b}m + 3m + k)\}) = (S - S) + (E^{-} \cup \{\mathfrak{c} - (\mathfrak{b}k - 4m)\})$$

Then the tracks $3m + k$ and $\mathfrak{b}k$ of S have the same content. If this holds for all $k = 1, 2, \dots, m$, then $S = \pi(S_1, \dots, S_{m-1})$ for some sets $S_i \subseteq \mathbb{N}$.

Proof. The sides of the equations are

$$\bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(T_1) = \bigcup_{i \neq \mathfrak{c}} \tau_i^{\mathfrak{p}}(\mathbb{N}) \cup \tau_{\mathfrak{c}}^{\mathfrak{p}}(T_2),$$

where T_1 has the same contents as track $\mathfrak{b}m + 3m + k$ of $S + S$ and T_2 has the same contents as track $\mathfrak{b}k - 4m$ of $S - S$. It follows that the contents on these tracks are the same, that is $T_1 = T_2$.

The set T_1 is equal to the sum of contents on tracks $\mathfrak{b}m$ and $3m + k$ of S. And since track $\mathfrak{b}m$ contains the set $\{0\}$, this sum equals the contents on track $3m + k$ of S.

Similarly, T_2 equals the subtraction of content on track $4m$ from content on track $\mathfrak{b}k$ of S. Again the track $4m$ contains $\{0\}$, and thus T_2 equals the content on track $\mathfrak{b}k$. It follows that the tracks $3m + k$ and $\mathfrak{b}k$ of S have the same content.

If this holds for all k, then $S = \pi(S_1, \ldots, S_{m-1})$ for some sets $S_i \subseteq \mathbb{N}$ by the definition of π. □

Weakly-Synchronized Ground Tree Rewriting (with Applications to Verifying Multithreaded Programs)

Anthony Widjaja Lin

Oxford University Department of Computer Science

Abstract. Ground tree rewrite systems (GTRS) are a well-known tree-extension of prefix-rewrite systems on words (a.k.a. pushdown systems), where subtrees (instead of word prefixes) are rewritten. GTRS can model programs with unbounded recursion depth and thread-spawning, wherein the threads have a tree-shaped dependency graph. We consider the extension of GTRS with a finite (global) control unit for synchronizing among the active threads, a.k.a. state-extended GTRS (sGTRS). Since sGTRS is Turing-complete, we restrict the finite control unit to dags possibly with self-loops, a.k.a. weakly-synchronized GTRS (wGTRS). wGTRS can be regarded as a generalization of context-bounded analysis of multipushdown systems with dynamic thread spawning. We show that reachability, repeated reachability, and the complement of model checking deterministic LTL over weakly-synchronized GTRS (wGTRS) are NP-complete by a polynomial reduction to checking existential Presburger formulas, for which highly optimized solvers are available.

1 Introduction

Pushdown systems (PDS) are a natural abstraction of sequential programs with unbounded recursions. Their verification problems have been extensively studied (e.g. see the survey [5]), many of which are not only decidable, but also relatively tractable.

Apart from having function calls, real-world programs are often multithreaded. Given the rapidly increasing popularity of multi-core computers, multithreaded applications are only becoming increasingly more popular. Most popular programming languages (e.g. Java, Python, C++, C#) now have built-in constructs to support multithreading, e.g., `Fork/Join`, `parbegin-parend`, `Parallel.For`. Such constructs allow an unbounded number of threads at any given time (due to thread-spawning). This motivates the study of verification problems on extensions of pushdown systems with multithreading.

In this paper, we start with a well-known extension of PDS called ground tree rewrite systems (GTRS), e.g., see [14]. Since PDS can be thought of as prefix-rewrite systems (prefixes are rewritten based on a given set of rewrite rules on words), GTRS can be construed as a tree-extension of PDS, wherein subtrees (instead of prefixes) are rewritten based on a given set of rewrite rules on ranked trees. Owing to the tree structure of GTRS, one can easily mimic the effect of `parbegin-parend` and `Parallel.For` language constructs, whereby a

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 630–642, 2012.

parent thread spawns several child threads and waits for the return values of computation of these child threads. This statement actually only holds *so long as there is no "shared" (global) variables*, which permit synchronization between the active (instead of waiting) threads.

A natural way to extend GTRS so as to allow synchronization via shared variables between the active threads is to extend GTRS with a finite number of (global) control states. Such an extension is called state-extended GTRS (sGTRS). Unlike GTRS, for which reachability and repeated reachability are solvable in polynomial time, sGTRS can easily simulate multistack pushdown automata for which most verification problems quickly become undecidable [19].

One way to extend GTRS with control states while staying within the realm of decidability is to disallow cycles (other than self-loops) in the transition graph of the control states of sGTRS. Such transition graphs of (with initial/accepting states) are often called 1-weak automata [16]. The class of sGTRS which satisfies this restriction is called weakly-synchronized (or weakly-extended) GTRS, which we abbreviate as wGTRS [21]. It is known [21] that reachability, repeated reachability, and the complement of model checking deterministic fragment of LTL ($\mathrm{LTL}_{\mathrm{det}}$) over wGTRS are all solvable in exponential time, but are NP-hard. LTL model checking over GTRS is known to be undecidable (e.g. see [4,11]). For GTRS, reachability and repeated reachability are in P (e.g. [14]).

What Is the Modeling Power of wGTRS? Useful timing and event constraints can be embedded in 1-weak automata (e.g. see [9]). In addition, we shall see later that wGTRS: (1) generalizes multipushdown systems with bounded context switches [18] by allowing dynamic thread creation using `parbegin-parend` or `Parallel.For` constructs, and (2) provides a natural underapproximation of sGTRS, where global synchronizations take place for at most a given bound n of times.

Contributions. The main contribution of this paper is a technique for showing optimal complexity of model checking weakly-synchronized GTRS by a poly-time reduction to satisfiability of existential Presburger formulas, which is NP-complete and for which there are highly-optimized solvers. We firstly consider the global reachability problem: given a wGTRS $\mathcal{P}$ over ranked alphabet Σ and two tuples $(s_0, \mathcal{S})$, $(t_0, \mathcal{T})$ of control states of $\mathcal{P}$ and tree automata over Σ, decide if there exists a path from some configuration (s_0, T_1) of $\mathcal{P}$, where $T_1 \in \mathcal{L}(\mathcal{S})$ to some configuration (t_0, T_2) of $\mathcal{P}$, where $T_2 \in \mathcal{L}(\mathcal{T})$. We show in Section 4 that this problem is NP-complete by a reduction to satisfiability of existential Presburger formulas.

We give several further applications of this upper bound in Section 5: (1) another poly-time algorithm for global reachability for GTRS and (2) NP-completeness for repeated reachability and the complement of $\mathrm{LTL}_{\mathrm{det}}$ model checking for wGTRS.

In the sequel, when deriving upper bounds, we allow infinitely many rewrite rules in the input (w)GTRS compactly represented by means of tree automata.

Other Related Work. There are two other approaches to extend pushdown systems with dynamic thread spawning. Process rewrite systems hierarchy proposed by Mayr [17]. Some classes of systems in this hierarchy (including PA and PAD processes) are intimately connected to GTRS [11]. Another approach was considered by Bouajjani *et al.* [3] in their work on networks of pushdown systems (called CPDN).

The authors of [13] studied the extension of process rewrite systems and other classes in the hierarchy with 1-weak finite control unit. They showed that decidability can still be retained for reachability, among others. Decidability and undecidability of fragments of LTL have also been fully classified [4]. The techniques considered in this paper can be easily adapted to show that reachability, repeated reachability for weakly extended PA and PAD are NP-complete, while LTL_{det} model checking for weakly extended PA and PAD are coNP-complete.

Context-bounded model checking over multipushdown systems was first studied in [18] and is shown to be NP-complete for multipushdown systems. Various extensions have been proposed including phase-bounds [22], ordered multi-stack machines [1], bounded languages [8,10], dynamic thread creation [2], and more general approach [15]. The work of [2] considers a different style of multithreading than what we consider in this paper. The difference is akin to the difference between GTRS and CPDN, which are unexplored. We leave it as future work to explore the connections.

2 Preliminaries

General Notations. For two given natural numbers $i \leq j$, we define $[i, j] = \{i, i+1, \ldots, j\}$. Define $[k] = [0, k]$. Given a function $f : S_1 \to S_2$ and a subset $S_1' \subseteq S_1$, the notation $f_{|_{S_1'}}$ is used to denote the restriction of f to the domain S_1'. Vectors $\mathbf{v}$ over a set S are simply elements of S^k for some positive integer k. An example is when $S = \mathbb{N}$, which gives us vectors of naturals. In the sequel, vectors are also thought of as a function $\mathbf{v} : I \to S$, where I is some nonempty finite index set. Vectors in the standard sense use $I = [1, k]$ for some positive integer k. When comparing two vectors of naturals $\mathbf{u}, \mathbf{v}$ over the same index set I, we use the component-wise ordering. We write $\mathbf{u} \leq \mathbf{v}$ iff, for each $i \in I$, $\mathbf{u}(i) \leq \mathbf{v}(i)$. This partial ordering $\leq$ is well-known to be well-founded.

Transition Systems. Let ACT be a finite set of *action symbols.* A *transition system* over ACT is a tuple $\mathfrak{S} = \langle S, \{\to_a\}_{a \in \mathsf{ACT}} \rangle$, where S is a set of *configurations*, and $\to_a \subseteq S \times S$ is a binary relation over S. We use $\to$ to denote the relation $\left(\bigcup_{a \in \mathsf{ACT}} \to_a\right)$. The notation $\to^+$ (resp. $\to^*$) is used to denote the transitive (resp. transitive-reflexive) closure of $\to$. Given two sets $S_1, S_2 \subseteq S$ of configurations, we write $S_1 \to^+ S_2$ if $s_1 \to^+ s_2$ for some $s_1 \in S_1$ and $s_2 \in S_2$. The notations $S_1 \to^* S_2$ and $S_1 \to S_2$ are defined likewise. We say that a sequence $s_1 \to \cdots \to s_n$ is a *path* (or *run*) in $\mathfrak{S}$ (or in $\to$). Given two paths $\pi_1 : s_1 \to^* s_2$ and $\pi_2 : s_2 \to^* s_3$ in $\to$, we may concatenate them to obtain $\pi_1 \odot \pi_2$ (by gluing together s_2). Given a subset $S' \subseteq S$, denote by $\textsc{Rec}^{\to}(S')$ to be the set of elements $s_0 \in S$ for which there exists an infinite path $s_0 \to s_1 \to \cdots$ visiting S' infinitely often, i.e., $s_j \in S'$ for infinitely many $j \in \mathbb{N}$. A transition system

$\mathfrak{S} = \langle S, \{\to_a\}_{a \in \mathsf{ACT}} \rangle$ is said to be *1-weak* if each path $s_1 \to \cdots \to s_n$ in $\mathfrak{S}$ with $s_1 = s_n$ satisfies $s_i = s_{i+1}$ for all $i \in [1, n-1]$. In other words, every cycle in $\mathfrak{S}$ is a self-loop.

Word Languages and Parikh Images. An alphabet Σ is a finite set of symbols. We use standard notations from word language theory (e.g. [12]). Given a word $w \in \Sigma^*$ and $a \in \Sigma$, we use $|w|_a$ to denote the number of occurrences of a in w (e.g. $|abaa|_a = 3$). The *Parikh image of* w, denoted by $\mathbb{P}(w)$, is the integral vector $\mathbf{v} : \Sigma \to \mathbb{N}$ such that $\mathbf{v}(a) = |w|_a$. Given a language $\mathcal{L} \subseteq \Sigma^*$, its *Parikh image* is $\mathbb{P}(\mathcal{L}) = \{\mathbb{P}(w) : w \in \mathcal{L}\}$.

Tree Automata and Languages. A *ranked alphabet* is a nonempty finite set of symbols Σ equipped with a rank function $\texttt{rank} : \Sigma \to \mathbb{N}$. When the context is clear, a ranked alphabet will simply be referred to as an alphabet. Let $\texttt{rank}(\Sigma)$ denote $\max\{\texttt{rank}(a) : a \in \Sigma\}$. A *tree domain* D is a nonempty finite subset of $\mathbb{N}^*$ satisfying (1) *prefix closure*, i.e., if $vi \in D$ with $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $v \in D$, (2) *younger-sibling closure*, i.e., if $vi \in D$ with $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $vj \in D$ for each natural number $j < i$. Standard terminologies (e.g. nodes, parents, children, ancestors, descendants) will be used. A *tree* over a ranked alphabet Σ is a pair $T = (D, \lambda)$, where D is a tree domain and the *node-labeling* λ is a function mapping D to Σ such that, for each node $v \in D$, the number of children of v in D equals the rank $\texttt{rank}(\lambda(v))$ of the node label of v. Write $\textsc{Tree}(\Sigma)$ for the set of all trees over Σ. In the sequel, we also use the standard term representations of trees (cf. [6]).

A (bottom-up) nondeterministic tree-automaton (NTA) over a ranked alphabet Σ is a tuple $\mathcal{A} = \langle Q, \Delta, F \rangle$, where Q is a finite nonempty set of states, Δ is a finite set of rules of the form $(q_1, \ldots, q_r) \overset{a}{\hookrightarrow} q$, where $a \in \Sigma$, $r = \texttt{rank}(a)$, and $q, q_1, \ldots, q_r \in Q$, and $F \subseteq Q$ is a set of final states. A rule of the form $() \overset{a}{\hookrightarrow} q$ is also written as $\overset{a}{\hookrightarrow} q$. For a state $q \in Q$, the notation $\mathcal{A}^q$ is used to denote the NTA $\langle Q, \Delta, \{q\} \rangle$. A *run* of $\mathcal{A}$ on a tree $T = (D, \lambda)$ is a mapping ρ from D to Q such that, for each node $v \in D$ (with label $a = \lambda(v)$) with its all children $v_1, \ldots, v_r$, it is the case that $(\lambda(v_1), \ldots, \lambda(v_r)) \overset{a}{\hookrightarrow} \lambda(v)$ is a transition in Δ. For a subset $Q' \subseteq Q$, the run is said to be *accepting* if $\rho(\epsilon) \in F$. The NTA is said to *accept* T if it has an accepting run on T. The *language* $\mathcal{L}(\mathcal{A})$ *of* $\mathcal{A}$ is the set of trees which are accepted by $\mathcal{A}$. A language L is said to be *regular* if there exists an NTA accepting L.

A *context tree* with *(context) variables* $x_1, \ldots, x_n$ is a tree $T = (D, \lambda)$ over the alphabet $\Sigma \cup \{x_1, \ldots, x_n\}$, where $\Sigma \cap \{x_1, \ldots, x_n\} = \emptyset$ and for each $i = 1, \ldots, n$, it is the case that $\texttt{rank}(x_i) = 0$ and there exists a unique *context node* u_i with $\lambda(u_i) = x_i$. In the sequel, we will often denote such a context tree as $T[x_1, \ldots, x_n]$ and, by convention, assume that $u_1, \ldots, u_n$ appear in an inorder tree traversal ordering. Given trees $T_1 = (D_1, \lambda_1), \ldots, T_n = (D_n, \lambda_n)$ over Σ, we use the notation $T[T_1, \ldots, T_n]$ to denote the tree (D', λ') obtained by filling each hole x_i by T_i, i.e., $D' = D \cup \bigcup_{i=1}^{n} u_i \cdot D_i$ and $\lambda'(u_i v) = \lambda_i(v)$ for each $i = 1, \ldots, n$ and $v \in D_i$. Given a tree T, if $T = C[t]$ for some context tree $C[x]$ and a tree t, then t is called a *subtree* of T.

Notation for Context-Free Grammars. A *context-free grammar* (CFG) over an alphabet Σ is a tuple $\mathcal{G} = (\mathtt{Nt}, \mathtt{Tt}, \mathtt{Rules}, \mathtt{Start})$, where $\mathtt{Nt}$ is a finite set of *nonterminals*, $\mathtt{Tt} = \Sigma$ is a finite set *terminals*, $\mathtt{Rules}$ is a finite set of *production rules* of the form $X \to \alpha$ where $X \in \mathtt{Nt}$ and $\alpha \in (\mathtt{Nt} \cup \mathtt{Tt})^*$, and $\mathtt{Start} \in \mathtt{Nt}$ is the *start nonterminal.* In the sequel, for each $X \in \mathtt{Nt}$, we use the notation $\mathcal{G}^X$ to denote the CFG $(\mathtt{Nt}, \mathtt{Tt}, \mathtt{Rules}, X)$. We denote by $\mathcal{L}(\mathcal{G})$ the language of words generated by $\mathcal{G}$.

Existential Presburger Formulas. Existential Presburger formulas are formulas in the existential fragment of Presburger arithmetic, i.e., first-order theory over $\langle \mathbb{N}, + \rangle$. If $\varphi(\mathbf{x})$ is a formula with the vector $\mathbf{x}$ of free variables, where $\mathbf{x} : I \to \{x_1, \ldots, x_m\}$ is a vector with some index set I, and $\mathbf{v} : I \to \mathbb{N}$ is a vector over natural numbers, we write $\langle \mathbb{N}, + \rangle \models \varphi(\mathbf{v})$ if φ is a true formula in $\langle \mathbb{N}, + \rangle$ under the interpretation that maps each variable $\mathbf{x}(i)$ to $\mathbf{v}(i)$. A formula $\varphi(\mathbf{x})$ is said to be *satisfiable in* $\langle \mathbb{N}, + \rangle$ if there exists $\mathbf{v}$ such that $\langle \mathbb{N}, + \rangle \models \varphi(\mathbf{v})$. It is well-known that deciding satisfiability over $\langle \mathbb{N}, + \rangle$ is NP-complete [20], for which there are highly optimized solvers (e.g. Z3 [7]).

3 Weakly Extended Ground Tree Rewrite Systems

A *state-extended ground tree rewrite systems (sGTRS)* over a finite set ACT of action symbols is a tuple $\mathcal{P} = \langle Q, \Sigma, \Delta \rangle$, where Q is a nonempty finite set of control states, Σ is a ranked alphabet, and Δ is a finite set of rules of the form $(q_1, \mathcal{A}_1) \overset{\alpha}{\hookrightarrow} (q_2, \mathcal{A}_2)$, where $q_1, q_2 \in Q$, $\alpha \in \mathsf{ACT}$, and $\mathcal{A}_1, \mathcal{A}_2$ are NTA over Σ. A configuration of $\mathcal{P}$ is a tuple (q, T), where $q \in Q$ and $T \in \textsc{Tree}(\Sigma)$. Let $\mathrm{Conf}(\mathcal{P})$ denote the set of configurations of $\mathcal{P}$. The transition system generated by $\mathcal{P}$ is $\mathfrak{S}_{\mathcal{P}} = \langle \mathrm{Conf}(\mathcal{P}), \{\to_a\}_{a \in \mathsf{ACT}} \rangle$, where $(q_1, T_1) \to_\alpha (q_2, T_2)$ iff there exist a context tree $T[x]$, a rule $(q_1, \mathcal{A}_1) \overset{\alpha}{\hookrightarrow} (q_2, \mathcal{A}_2)$ in Δ, and trees $t_1 \in \mathcal{L}(\mathcal{A}_1)$ and $t_2 \in \mathcal{L}(\mathcal{A}_2)$ such that $T_1 = T[t_1]$ and $T_2 = T[t_2]$. We define $\to_{\mathcal{P}}$ to be the union of all $\to_a$, where a ranges over ACT.

The *underlying control graph* of $\mathcal{P} = \langle Q, \Sigma, \Delta \rangle$ is the finite transition system $\mathfrak{S} = \langle Q, \{\to_a\}_{a \in \mathsf{ACT}} \rangle$, where $q_1 \to_a q_2$ iff $(q_1, \mathcal{A}_1) \overset{a}{\hookrightarrow} (q_2, \mathcal{A}_2)$ is a rule in Δ. We say that $\mathcal{P}$ is a *weakly-synchronized (or weakly-extended) ground tree rewrite systems (wGTRS)* if its underlying control graph is 1-weak. In the case of wGTRS, we often denote edge relation of this underlying control graph as $\prec$. We say that $\mathcal{P}$ is a *ground tree rewrite systems (GTRS)* if its underlying control graph is $\langle \{q\}, \{\to_a\}_{a \in \mathsf{ACT}} \rangle$, where $q \to_a q$, for each $a \in \mathsf{ACT}$. In this case, a GTRS $\mathcal{P} = \langle Q, \Sigma, \Delta \rangle$ is also written as $\langle \Sigma, \Delta \rangle$ (or simply Δ) for simplicity. If each letter in Σ is of rank ≤ 1, $\mathcal{P}$ is also called a *pushdown system (PDS).*

Remark: "Ground tree rewrite systems" are often defined in a rather restrictive form, wherein each NTA $\mathcal{A}$ in the rewrite rule is explicitly given as a tree $t \in \textsc{Tree}(\Sigma)$ representing the singleton set $\{t\}$. Our definition of GTRS coincides with what is commonly referred to as "regular GTRS". See [14].

Notation: In the sequel, given $s \in Q$ and an NTA $\mathcal{A}$ over Σ, we shall use the notation $(s, \mathcal{L}(\mathcal{A}))$ to mean $\{s\} \times \mathcal{L}(\mathcal{A})$.

We define the *(global) reachability problem for sGTRSs* as follows: given two NTAs $\mathcal{A}_1, \mathcal{A}_2$ over the ranked alphabet Σ, an sGTRS $\mathcal{P} = \langle Q, \Sigma, \Delta \rangle$, and two states $q_1, q_2 \in Q$, decide if $(q_1, \mathcal{L}(\mathcal{A}_1)) \rightarrow^*_{\mathcal{P}} (q_2, \mathcal{L}(\mathcal{A}_2))$. When we restrict the input sGTRSs to wGTRSs (resp. GTRSs), we call the resulting subproblems to be *global reachability problem for wGTRSs (resp. GTRSs).*

An Intuitive Example. Modeling sequential programs as PDS is standard (e.g. see [17]): the stack is used to record program points (function names and values of local variables), while the finite control is used to record the return values from the previous function call. Modeling with (s)GTRS is similar except that the tree structure can model multithreading.

Consider a program with two functions called `fun1` and `fun2` (among others). The function `fun2`, which we leave unspecified, inputs and outputs a boolean value. The function `fun1` is defined in as follows:

$$\texttt{bool b[5]; Par.For(0,4,i,=>)}\{\texttt{b[i] = fun2(a[i])}\}\texttt{;return} \bigwedge_{i=0}^{4} \texttt{b[i];}$$

In this example, we assume that the boolean array `a` is given as input to `fun1` and has size 5. This program simply executes the assignment `b[i] = fun2(a[i])` in parallel for each $i \in [0, 4]$, and afterwards outputs the boolean value obtained by taking the conjunction of all the boolean variables `b[i]`.

Assuming there is no global variables, the above program can be easily modeled as a GTRS. For example, a GTRS model may contain the following rules: (1) $\langle \texttt{fun1}, \mathrm{s}\ 2, a \rangle \rightarrow \langle \texttt{fun1}, \mathrm{s}\ 3, a \rangle (\langle \texttt{fun2}, \mathrm{s}\ 1, a[0] \rangle, \ldots, \langle \texttt{fun2}, \mathrm{s}\ 1, a[4] \rangle)$, reflecting the `Par.For` step (s i means step i), and (2) for all $j_1, \ldots, j_4 \in \{0, 1\}$, $\langle \texttt{fun1}, \mathrm{s}\ 3, a \rangle (j_1, \ldots, j_4) \rightarrow \bigwedge_{i=0}^{4} j_i$, reflecting the return step of `fun1`.

With the existence of global (shared) variables, the above GTRS does not suffice because after rule of type (1) has been applied, the five subthreads can no longer communicate in this GTRS. Communication in general can be captured by sGTRS by embedding synchronization in the finite control. wGTRS actually suffices provided that the vector of values of the shared variables can change only for a bounded number of times.

Modeling Power of wGTRS. wGTRS can be used to underapproximate sGTRS. Intuitively, given an sGTRS $\mathcal{P}$ and a "depth" parameter $d \in \mathbb{N}$, a wGTRS $\mathcal{P}_d$ is constructed in polynomial time that underapproximates $\mathcal{P}$ up to d switches of control states. We can also show that context-bounded analysis of multipushdown systems [18] can be efficiently reduced to analyzing wGTRS. Both are shown in the full version.

4 Reachability

Theorem 1. *Global reachability for wGTRS is NP-complete. In fact, it is polytime reducible to satisfiability of existential Presburger formulas.*

NP-hardness follows from the proof of Proposition 5.4.6 in [21] (by a reduction from hamiltonian path problem). We now show the upper bound. The idea of the reduction to satisfiability of existential Presburger formulas is as follows: first construct a CFG $\mathcal{G}$ which "overapproximates" the given wGTRS $\mathcal{P}$; the behavior of $\mathcal{G}$ is then limited by adding an extra existential Presburger constraint ψ. Since there is a linear-time algorithm [24] for computing the Parikh image of $\mathcal{L}(\mathcal{P})$ as existential Presburger formulas Ψ, the desired formula will be $\Psi \wedge \psi$.

We now provide the details of the reduction. We are given a wGTRS $\mathcal{P} = \langle Q, \Sigma, \Delta \rangle$ over the action alphabet ACT, and two tuples $(s_0, \mathcal{S})$ and $(t_0, \mathcal{T})$ of states $s_0, t_0 \in Q$ and NTA $\mathcal{S}, \mathcal{T}$ over Σ representing, respectively, a set $\{(s_0, T) : T \in \mathcal{L}(\mathcal{S})\}$ of start configurations and a set $\{(t_0, T) : T \in \mathcal{L}(\mathcal{T})\}$ of target configurations. Denote by $\mathfrak{S}_{\mathcal{P}} = \langle \mathrm{Conf}(\mathcal{P}), \{\to_a\}_{a \in \mathsf{ACT}} \rangle$ the transition system generated by $\mathcal{P}$. The task is to decide whether $(s_0, \mathcal{L}(\mathcal{S})) \to^* (t_0, \mathcal{L}(\mathcal{T}))$.

Denote the a-labeled edge relation of the underlying control graph G of $\mathcal{P}$ by $\prec_a$, and the transitive closure (resp. transitive-reflexive closure) of $\prec := \bigcup_{a \in \mathsf{ACT}} \prec_a$ by $\prec^+$ (resp. $\prec^*$). Since G is a DAG possibly with self-loops, it follows that $\prec^+$ is antisymmetric, i.e., if $s_1 \prec^+ s_2$ and $s_2 \prec^+ s_1$, then $s_1 = s_2$. *Without loss of generality, we assume that: (1)* $s_0 \prec^* t_0$ (for, otherwise, we immediately have $(s_0, \mathcal{L}(\mathcal{S})) \not\to^* (t_0, \mathcal{L}(\mathcal{T}))$), and (2) each state $s \in Q$ satisfy $s_0 \prec^* s \prec^* t_0$ (for all $T, T' \in \textsc{Tree}(\Sigma)$, i.e., each path $(s_0, T) \to^* (t_0, T')$ cannot go via configurations of the form (s, T'') with either $s_0 \not\prec^* s$ or $s \not\prec^* t_0$).

We now define the CFG $\mathcal{G} = (\mathtt{Nt}, \mathtt{Tt}, \mathtt{Rules}, \mathtt{Start})$. In the following, we use the notation $\mathcal{M}$ (possibly with a subscript) to range over the NTAs $\mathcal{S}$, $\mathcal{T}$, or NTAs appearing in Δ. The notation $q^{\mathcal{M}}$ (possibly with a subscript) will be used to denote a state in the NTA $\mathcal{M}$. The notation $q_F^{\mathcal{M}}$ will be used to denote a final state of $\mathcal{M}$. The starting nonterminal $\mathtt{Start} \in \mathtt{Nt}$ is marked. Add the rule $\mathtt{Start} \to X_{(s_0, q_F^{\mathcal{S}}), (t_0, q_F^{\mathcal{T}})}$, for each $q_F^{\mathcal{S}}$ and each $q_F^{\mathcal{T}}$. The nonterminal $X_{(s_0, q_F^{\mathcal{S}}), (t_0, q_F^{\mathcal{T}})}$ is initially unmarked. We then repeat the following two rules until all elements of $\mathtt{Nt}$ have been marked. If $X_{(s, q^{\mathcal{M}_1}), (t, q^{\mathcal{M}_2})} \in \mathtt{Nt}$ is unmarked, then mark $X_{(s, q^{\mathcal{M}_1}), (t, q^{\mathcal{M}_2})}$ and apply the following rules:

(Rule I) For all transitions $(q_1^{\mathcal{M}_1}, \ldots, q_r^{\mathcal{M}_1}) \overset{a}{\hookrightarrow} q^{\mathcal{M}_1}$ and $(q_1^{\mathcal{M}_2}, \ldots, q_r^{\mathcal{M}_2}) \overset{a}{\hookrightarrow} q^{\mathcal{M}_2}$ in $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively, add the rule

$$X_{(s, q^{\mathcal{M}_1}), (t, q^{\mathcal{M}_2})} \quad \to \quad X_{(s, q_1^{\mathcal{M}_1}), (t, q_1^{\mathcal{M}_2})} \cdots X_{(s, q_r^{\mathcal{M}_1}), (t, q_r^{\mathcal{M}_2})}$$

to $\mathtt{Rules}$ and add each $X_{(s, q_i^{\mathcal{M}_1}), (t, q_i^{\mathcal{M}_2})}$ on the r.h.s. of the rule to $\mathtt{Nt}$ unmarked (if not already a member of $\mathtt{Nt}$).

(Rule II) For each wGTRS rule $r = (s', \mathcal{A}) \overset{\alpha}{\hookrightarrow} (t', \mathcal{B})$ with $s \prec^* s' \prec t' \prec^* t$, and each $q_F^{\mathcal{A}}$ and each $q_F^{\mathcal{B}}$, add the rule

$$X_{(s, q^{\mathcal{M}_1}), (t, q^{\mathcal{M}_2})} \quad \to \quad \alpha(s', t') X_{(s, q^{\mathcal{M}_1}), (s', q_F^{\mathcal{A}})} X_{(t', q_F^{\mathcal{B}}), (t, q^{\mathcal{M}_2})}$$

to $\mathtt{Rules}$, add $X_{(s, q^{\mathcal{M}_1}), (s', q_F^{\mathcal{A}})}$ and $X_{(t', q_F^{\mathcal{B}}), (t, q^{\mathcal{M}_2})}$ to $\mathtt{Nt}$ unmarked (if not already a member of $\mathtt{Nt}$), and add (s', t') and each letter in $\alpha \in \mathsf{ACT}^*$ to $\mathtt{Tt}$.

Observe that $s \prec^+ t$ for each $X_{(s, q^{\mathcal{M}_1}), (t, q^{\mathcal{M}_2})} \in \mathtt{Nt}$ is an invariant throughout the above procedure. Termination of this procedure is immediate since (1) $\mathtt{Nt} \subseteq \bigcup_{\mathcal{M}_1, \mathcal{M}_2} \{X_{(s, q^{\mathcal{M}_1}), (t, q^{\mathcal{M}_2})} : s \prec^+$

t, and for each $i = 1, 2$, $q^{\mathcal{M}_i}$ is a state of $\mathcal{M}_i\}$ and the size of the set on the r.h.s. is at most $|Q|^2 \times$ (total number of NTA states in $\mathcal{P})^2$, and (2) the r.h.s. of each production rule is a word of length at most $\max\{\mathtt{rank}(\Sigma), 4\}$.

Let $m := |\mathtt{Tt}|$. We use the linear-time algorithm from [24] on the input $\mathcal{G}$ to produce an existential Presburger formula $\Psi(\mathbf{x})$, where $\mathbf{x} : \mathtt{Tt} \to \{x_1, \ldots, x_m\}$, capturing the Parikh image of $\mathcal{L}(\mathcal{G})$, i.e., for each $\mathbf{v} : \mathtt{Tt} \to \mathbb{N}$, we have $\mathbf{v} \in \mathbb{P}(\mathcal{L}(\mathcal{G}))$ iff $\langle \mathbb{N}, + \rangle \models \Psi(\mathbf{v})$. We also write $\mathbf{x}(s,t)$ to mean $\mathbf{x}((s,t))$, if $(s,t) \in \mathtt{Tt}$.

We now define several constraints as quantifier-free Presburger formulas:

- $\textsc{Dom} := \bigwedge_{s \prec t, s \neq t} \mathbf{x}(s,t) \leq 1$. This "domain" formula asserts that advancing from control state s to its strict successor t can take place at most once.
- $\textsc{Out}_{\leq 1} := \bigwedge_{s \prec t_1, s \prec t_2, \text{distinct}(s,t_1,t_2)} (\mathbf{x}(s,t_1) = 1 \to \mathbf{x}(s,t_2) = 0)$. This formula asserts that from control state s, the system can only advance to *at most one* of its successors.
- $\textsc{NoIncomp} := \bigwedge_{s : s_0 \prec^+ s \prec^+ t_0} \left(\textsc{Inv}_s \to \bigwedge_{s' : s' \neq s, s \not\prec^+ s', s' \not\prec^+ s} \neg \textsc{Inv}_{s'} \right)$, where $\textsc{Inv}_t$ is a shorthand for the formula $\bigvee_{s \prec t} \mathbf{x}(s,t) = 1 \vee \bigvee_{t \prec s} \mathbf{x}(t,s) = 1$. Intuitively, if a control state s is involved in a path, then no control states that are incomparable to s (with respect to $\prec^+$) can be involved in this path.
- if $s_0 = t_0$, then $\textsc{Init} := \top$, and if $s_0 \neq t_0$, then $\textsc{Init} := \bigvee_{s_0 \prec t, s_0 \neq t} \mathbf{x}(s_0, t) = 1$. This formula states that the system must advance from the initial state.
- $\textsc{Progress} := \bigwedge_{t \in Q, s_0 \prec^+ t \prec^+ t_0, \text{distinct}(s_0,t,t_0)}$ $\left(\bigvee_{s \prec t, s \neq t} \mathbf{x}(s,t) = 1 \to \bigvee_{t \prec t', t \neq t'} \mathbf{x}(t,t') = 1 \right)$. This formula states that the system must advance from a control state it is in to one of its successors.

In the sequel, we let φ_1 denote the formula $\textsc{Dom} \wedge \textsc{Out}_{\leq 1} \wedge \textsc{NoIncomp}$, and φ_2 denote the formula $\textsc{Init} \wedge \textsc{Progress}$. The desired existential Presburger formula is $\varphi := \Psi \wedge \varphi_1 \wedge \varphi_2$. Correctness of the reduction follows from the following lemma.

Lemma 2 (Correctness of Reduction). *For each $w \in \mathsf{ACT}^*$, $(s_0, \mathcal{L}(\mathcal{S})) \to_v (t_0, \mathcal{L}(\mathcal{T}))$ for some $v \in \mathsf{ACT}^*$ with $\mathbb{P}(v) = \mathbb{P}(w)$ iff there exists $\mathbf{v} : \mathtt{Tt} \to \mathbb{N}$ such that $\mathbf{v}(a) = |w|_a$ for each $a \in \mathsf{ACT}$ and $\langle \mathbb{N}, + \rangle \models \varphi(\mathbf{v})$.*

It is not hard to show that the reduction takes polynomial time; more precisely, $O(|Q|^3 + (\mathtt{rank}(\Sigma) \times N))$ time, where $N := (|Q|^2 \times N_{\max}^2) \times (M_{\max}^2 + |\Delta|)$, $N_{\max}$ be the maximum number of automata states in any given NTA appearing in $\mathcal{P}$ or $\mathcal{S}$ or $\mathcal{T}$, and $M_{\max}$ be the maximum number of automata transitions in any given NTA in $\mathcal{P}$ or $\mathcal{S}$ or $\mathcal{T}$. The analysis is given in the full version.

Remark: Adding a "counting constraint" on the path as an existential Presburger formula is easy. Such a counting constraint is simply an existential Presburger formula $\psi(\mathbf{x}')$, where $\mathbf{x}' = \mathbf{x}_{|\mathsf{ACT}}$. In this case, the desired formula is simply $\varphi \wedge \psi$.

Correctness: Proof of Lemma 2. The proofs for both directions of Lemma 2 are done by induction. However, in both cases, we will have to strengthen the statements; for, otherwise, the induction hypothesis will not get us off the ground. To this end, we will define a slight variant of the transition system $\mathfrak{S}_{\mathcal{P}}$ generated by $\mathcal{P}$ (which we call $\mathfrak{S}'_{\mathcal{P}}$).

Let $\mathfrak{S}'_{\mathcal{P}}$ be the transition system obtained by adding to $\mathfrak{S}_{\mathcal{P}}$ each ϵ-transition $(s,T) \to_\epsilon (t,T)$, for each $T \in \textsc{Tree}(\Sigma)$ and $s \prec^+ t$. Given a path

$$\pi = (p_0, T_0) \to_{a_1} \cdots \to_{a_n} (p_n, T_n)$$

in $\mathfrak{S}'_{\mathcal{P}}$ and $w = a_1 \cdots a_n$, we define $\chi(\pi)$ to be the vector $\mathbf{v} : \mathtt{Tt} \to \mathbb{N}$ such that (1) if $a \in \mathsf{ACT}$, then $\mathbf{v}(a) = |w|_a$, and (2) if $a = (s,t)$ with $s \prec t$, then $\mathbf{v}(a)$ is the number of indices $i \in [1,n]$ with $(p_{i-1}, p_i) = (s,t)$ and $a_i \neq \epsilon$.

Lemma 3. *For all $X_{(s,q^{\mathcal{M}_1}),(t,q^{\mathcal{M}_2})} \in \mathtt{Nt}$, if $w \in \mathcal{L}(\mathcal{G}^{X_{(s,q^{\mathcal{M}_1}),(t,q^{\mathcal{M}_2})}})$ with $\langle \mathbb{N}, + \rangle \models \varphi_1(\mathbb{P}(w))$, then there exists a path $\pi : (s, \mathcal{L}(\mathcal{M}_1^{q^{\mathcal{M}_1}})) \to_v (t, \mathcal{L}(\mathcal{M}_2^{q^{\mathcal{M}_2}}))$ in $\mathfrak{S}'_{\mathcal{P}}$ with $\chi(\pi) = \mathbb{P}(w)$.*

It is not hard to see that Lemma 3 implies the direction ($\Leftarrow$) of Lemma 2. A proof can be found in the full version.

Proof (of Lemma 3). As we previously saw, we have $s \prec^+ t$ for each $X_{(s,q^{\mathcal{M}_1}),(t,q^{\mathcal{M}_2})} \in \mathtt{Nt}$. The proof is by induction on the length of derivations of the word w from $X_{(s,q^{\mathcal{M}_1}),(t,q^{\mathcal{M}_2})}$.

The base case is when $X_{(s,q^{\mathcal{M}_1}),(t,q^{\mathcal{M}_2})} \to \epsilon$, which is a production rule generated by Rule I. This means that there exists $a \in \Sigma$ such that $\overset{a}{\hookrightarrow} q^{\mathcal{M}_1}$ and $\overset{a}{\hookrightarrow} q^{\mathcal{M}_2}$ are transitions of $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively, and thus $a \in \mathcal{L}(\mathcal{M}_1^{q^{\mathcal{M}_1}}) \cap \mathcal{L}(\mathcal{M}_2^{q^{\mathcal{M}_2}})$. Since $s \prec^+ t$, it follows that $(s,a) \to_\epsilon (t,a)$ is path π in $\mathfrak{S}'_{\mathcal{P}}$. It is also easy to see that $\mathbb{P}(\epsilon) = \chi(\pi) = \mathbf{0}$, and that $\langle \mathbb{N}, + \rangle \models \varphi_1(\mathbf{0})$.

We now proceed to the induction cases. There are two cases. We only consider the first case; the second case is considered in the full version.

The first induction case is when the first production rule applied in the derivation of w from $X_{(p_1,q^{\mathcal{M}_1}),(p_2,q^{\mathcal{M}_2})}$ (with $p_1 = s$ and $p_2 = t$) is

$$X_{(p_1,q^{\mathcal{M}_1}),(p_2,q^{\mathcal{M}_2})} \quad \to \quad \alpha(p_3,p_4) X_{(p_1,q^{\mathcal{M}_1}),(p_3,q^{\mathcal{M}_3})} X_{(p_4,q^{\mathcal{M}_4}),(p_2,q^{\mathcal{M}_2})}$$

where $q^{\mathcal{M}_3}$ and $q^{\mathcal{M}_4}$ are final states of $\mathcal{M}_3$ and $\mathcal{M}_4$, respectively. This production rule is generated by Rule II, which means that there exists a rule $r = (p_3, \mathcal{M}_3) \overset{\alpha}{\hookrightarrow} (p_4, \mathcal{M}_4)$ with $p_1 \prec^* p_3 \prec p_4 \prec^* p_2$ in $\mathcal{P}$. We may also write w as $\alpha(p_3,p_4) w_1 w_2$, where $w_1 \in \mathcal{L}(\mathcal{G}^{X_{(p_1,q^{\mathcal{M}_1}),(p_3,q^{\mathcal{M}_3})}})$ and $w_2 \in \mathcal{L}(\mathcal{G}^{X_{(p_4,q^{\mathcal{M}_4}),(p_2,q^{\mathcal{M}_2})}})$. Furthermore, since $\langle \mathbb{N}, + \rangle \models \varphi_1(\mathbb{P}(w))$ and $\mathbb{P}(w_1), \mathbb{P}(w_2) \leq \mathbb{P}(w)$, it follows that $\langle \mathbb{N}, + \rangle \models \varphi_1(\mathbb{P}(w_i))$ for each $i = 1, 2$. By induction, there exist paths $\pi_1 : (p_1, T_1) \to^* (p_3, T_3)$ and $\pi_2 : (p_4, T_4) \to^* (p_2, T_2)$ in $\mathfrak{S}'_{\mathcal{P}}$ such that $T_i \in \mathcal{L}(\mathcal{M}_i^{q^{\mathcal{M}_i}})$, for each $i \in [1,4]$, and $\chi(\pi_j) = \mathbb{P}(w_j)$, for each $j = 1, 2$. By applying the rule r above, we also see that $\pi_3 : (p_3, T_3) \to_\alpha (p_4, T_4)$ is a transition in $\mathfrak{S}_{\mathcal{P}}$. Therefore, $\pi := \pi_1 \odot \pi_3 \odot \pi_2$ is a path from (p_1, T_1) to (p_2, T_2) in $\mathfrak{S}'_{\mathcal{P}}$. We have $\chi(\pi) = \sum_{i=1}^{3} \chi(\pi_i) = \mathbb{P}(w_1) + \mathbb{P}(w_2) + \mathbb{P}(\alpha(p_3,p_4)) = \mathbb{P}(w)$. This completes the proof for the second induction case. □

It remains to prove the direction (⇒) of Lemma 2. To this end, we need the following lemma.

Lemma 4. *For all $X_{(s,q^{\mathcal{M}_1}),(t,q^{\mathcal{M}_2})} \in \mathtt{Nt}$, if there exists a path $\pi : (s, \mathcal{L}(\mathcal{M}_1^{q^{\mathcal{M}_1}})) \to_v (t, \mathcal{L}(\mathcal{M}_2^{q^{\mathcal{M}_2}}))$ in $\mathfrak{S}'_{\mathcal{P}}$, then there exists $w \in \mathtt{Tt}^*$ with $\mathbb{P}(w) = \chi(\pi)$ such that $w \in \mathcal{L}(\mathcal{G}^{X_{(s,q^{\mathcal{M}_1}),(t,q^{\mathcal{M}_2})}})$.*

To show the direction (⇒) of Lemma 2, first observe that each path $\pi : (s_0, \mathcal{L}(\mathcal{S})) \to_v (t_0, \mathcal{L}(\mathcal{T}))$ in $\mathfrak{S}_{\mathcal{P}}$ is also a path in $\mathfrak{S}'_{\mathcal{P}}$. By Lemma 4, there exists a word $w \in \mathcal{L}(\mathcal{G})$ with $\chi(\pi) = \mathbb{P}(w)$. Let $\mathbf{v} := \mathbb{P}(w)$. It follows that $\langle \mathbb{N}, + \rangle \models \Psi(\mathbf{v})$. It suffices to show that $\langle \mathbb{N}, + \rangle \models \varphi_1(\mathbf{v}) \wedge \varphi_2(\mathbf{v})$. If $s_0 = t_0$, it is easy to see that $\langle \mathbb{N}, + \rangle \models \varphi_1(\mathbf{v}) \wedge \varphi_2(\mathbf{v})$. Therefore, assume that $s_0 \neq t_0$. In this case, we take the projection of π on to its first component (i.e. control states), say, $p_0, \ldots, p_k$ such that $p_0 = s$ and $p_k = t$. By removing duplicates, we may assume that $p_0, \ldots, p_k$ are pairwise distinct control states. This means that for all distinct $p, p' \in Q$, we have $\mathbf{v}(p, p') = 1$ iff $p = p_{i-1}$ and p_i for some $i \in [1, k]$. Since $p_0, \ldots, p_k$ is a path in the underlying graph of $\mathcal{P}$, it is easy to check now that $\langle \mathbb{N}, + \rangle \models \varphi_1(\mathbf{v}) \wedge \varphi_2(\mathbf{v})$ by exhausting all the five subconjuncts in the formula $\varphi_1 \wedge \varphi_2$.

We now prove Lemma 4. To this end, we need the following technical lemma about path decompositions for wGTRS.

Lemma 5. *For each path $\pi : (p, T_1) \to_v (q, T_2)$ in $\mathfrak{S}'_{\mathcal{P}}$, there exists a context tree $C[x_1, \ldots, x_n]$ for some $n \in \mathbb{N}$ such that:*

1. *$T_1 = C[t_1, \ldots, t_n]$ for some trees $t_1, \ldots, t_n \in \textsc{Tree}(\Sigma)$,*
2. *$T_2 = C[t'_1, \ldots, t'_n]$ for some trees $t'_1, \ldots, t'_n \in \textsc{Tree}(\Sigma)$,*
3. *for each $i \in [1, n]$, there exists a path $\pi_i : (p, t_i) \to^* (p_i^1, t_i^1) \to_{\alpha_i} (p_i^2, t_i^2) \to^* (q, t'_i)$ in $\mathfrak{S}'_{\mathcal{P}}$ for some rewrite rule $(p_i^1, \mathcal{A}_i) \overset{\alpha_i}{\hookrightarrow} (p_i^2, \mathcal{B}_i)$ in $\mathcal{P}$ such that $t_i^1 \in \mathcal{L}(\mathcal{A}_i)$ and $t_i^2 \in \mathcal{L}(\mathcal{B}_i)$.*
4. *$\chi(\pi) = \sum_{i=1}^{n} \chi(\pi_i)$.*

It is not hard to see that Lemma 5 implies Lemma 4. The proof is done by induction on $\chi(\pi)$ (with componentwise ordering $\leq$). The above path decomposition lemma is used to prove the inductive case. The proof is not hard but tedious, and so is relegated into the full version. For space reasons, we also relegate the proof of Lemma 5 into the full version.

5 Applications

A Polynomial-Time Algorithm for GTRS Reachability. The reduction from the previous section can be easily modified to give another polynomial-time algorithm for GTRS global reachability. Given the GTRS $\mathcal{P}$ and two tuples $(s_0, \mathcal{S})$ and $(t_0, \mathcal{T})$ of control states and NTAs, consider the CFG $\mathcal{G}$ and formula $\varphi = \Psi \wedge \varphi_1 \wedge \varphi_2$ produced by the reduction. Observe that, in the case of GTRS, we have $\varphi_1 \wedge \varphi_2 \equiv \top$. Therefore, Lemma 2 implies that $(s_0, \mathcal{L}(\mathcal{S})) \to_{\mathcal{P}}^* (t_0, \mathcal{L}(\mathcal{T}))$ iff Ψ is a satisfiable Presburger formula iff $\mathcal{L}(\mathcal{G}) \neq \emptyset$. Therefore, we have reduced global GTRS reachability to language emptiness of CFG, which is solvable in polynomial time. This gives us another proof of the following proposition.

Proposition 6. *GTRS global reachability is solvable in polynomial-time.*

Repeated Reachability. *Repeated reachability for sGTRS* is the following problem: given an sGTRS $\mathcal{P} = \langle Q, \Sigma, \Delta \rangle$, an initial set $(s_0, \mathcal{L}(\mathcal{S}))$ of configurations of $\mathcal{P}$ given as $(s_0, \mathcal{S})$, a final set $\mathcal{C}$ of target configurations of $\mathcal{P}$ given as a function mapping a control state $p \in Q$ to an NTA $\mathcal{A}_p$ over Σ (here, $\mathcal{C}$ consists of configurations of the form (p, T) for some $p \in Q$ and $T \in \mathcal{L}(\mathcal{A}_p)$), decide whether $(s_0, \mathcal{L}(\mathcal{S})) \cap \textsc{Rec}^{\to_\mathcal{P}}(\mathcal{C}) \neq \emptyset$. As for reachability problem, this problem is undecidable.

Theorem 7. *Repeated reachability for wGTRS is NP-complete. In fact, it is poly-time reducible to satisfiablity of existential Presburger formulas.*

NP-hardness follows from the proof of Proposition 5.4.6 in [21] (by a reduction from hamiltonian path problem). To obtain the upper bound for this theorem, we reduce this problem to satisfiability of existential Presburger formulas in polynomial time. To this end, for each $p \in Q$, we define $\mathcal{P}_p$ to be GTRS obtained by restricting $\mathcal{P}$ to control state p, i.e., $\mathcal{P}_p = \langle \{p\}, \Sigma, \Delta_p \rangle$, where Δ_p consists of all rules in Δ of the form $(p, \mathcal{A}) \to_a (p, \mathcal{B})$. We first use Löding's result [14] that an NTA $\mathcal{A}'_p$ representing $\textsc{Rec}^{\to_{\mathcal{P}_p}}((p, \mathcal{L}(\mathcal{A}_p)))$ is computable in time polynomial in $\|\mathcal{A}_p\| + \|\mathcal{P}_p\|$, for each $p \in Q$. It follows that $(s_0, \mathcal{L}(\mathcal{S})) \cap \textsc{Rec}^{\to_\mathcal{P}}(\mathcal{C}) \neq \emptyset$ iff, for some $p \in Q$, $(s_0, \mathcal{L}(\mathcal{S})) \to^*_\mathcal{P} (p, \mathcal{L}(\mathcal{A}'_p))$. Applying our poly-time reduction from the previous section for the problem instance $(s_0, \mathcal{L}(\mathcal{S})) \to^*_\mathcal{P} (p, \mathcal{L}(\mathcal{A}'_p))$, for each p, and existentially quantifying all free variables, we obtained existential Presburger sentences φ_p such that $(s_0, \mathcal{L}(\mathcal{S})) \to^*_\mathcal{P} (p, \mathcal{L}(\mathcal{A}'_p))$ iff $\langle \mathbb{N}, + \rangle \models \varphi_p$. It immediately follows that $(s_0, \mathcal{L}(\mathcal{S})) \cap \textsc{Rec}^{\to_\mathcal{P}}(\mathcal{C}) \neq \emptyset$ iff $\langle \mathbb{N}, + \rangle \models \bigvee_{q \in Q} \varphi_p$. This shows that the desired existential Presburger sentence is $\bigvee_{q \in Q} \varphi_p$.

Model Checking Deterministic LTL over wGTRS. Deterministic LTL (LTL_{det}) over ACT is the fragment of LTL with the following syntax:

$$\varphi, \varphi' := p \mid \mathbf{X}\varphi \mid \varphi \wedge \varphi' \mid (p \wedge \varphi) \vee (\neg p \wedge \varphi') \mid (p \wedge \varphi)\mathbf{Op}(\neg p \wedge \varphi')$$

where p ranges over boolean combinations of ACT, and $\mathbf{Op}$ ranges over $\{\mathbf{U}, \mathbf{W}\}$. The semantics $[\![\varphi]\!] \subseteq \mathsf{ACT}^\omega$ of an LTL formula φ can be defined in the same way as for LTL, which is standard (e.g. see [23]). Given a transition system $\mathfrak{S} = \langle S, \{\to_a\}_{a \in \mathsf{ACT}} \rangle$ over ACT, we write $[\![\varphi]\!]_\mathfrak{S}$ to denote the set of configurations $s_0 \in S$ from which all infinite paths $\pi : s_0 \to_{a_1} s_1 \to_{a_2} \cdots$ in $\mathfrak{S}$ satisfy $a_1 a_2 \ldots \in [\![\varphi]\!]$. The problem of *model checking deterministic LTL for sGTRS* is defined as follows: given an LTL_{det} formula ψ and a sGTRS $\mathcal{P} = \langle Q, \Sigma, \Delta \rangle$ over the same set ACT of action symbols, and an initial set of configurations $(s_0, \mathcal{L}(\mathcal{S}))$ of $\mathcal{P}$ represented as the tuple $(s_0, \mathcal{S})$ of control state $s_0 \in Q$ and NTA $\mathcal{S}$ over Σ, decide if $(s_0, \mathcal{L}(\mathcal{S})) \subseteq [\![\psi]\!]_\mathfrak{S}$.

Theorem 8. LTL_{det} *model checking over wGTRS is coNP-complete. In fact, it is poly-time reducible to non-satisfiablity of existential Presburger formulas.*

coNP-hardness for the problem is known [21]. For space reasons, we relegate the proof for the upper bound into the full version.

Acknowledgment. I thank Matthew Hague and anonymous reviewers for their helpful feedback. I am grateful to EPSRC (EP/H026878/1) for their support.

References

1. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of Multi-pushdown Automata Is 2ETIME-Complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
2. Atig, M.F., Bouajjani, A., Qadeer, S.: Context-bounded analysis for concurrent programs with dynamic creation of threads. LMCS 7(4) (2011)
3. Bouajjani, A., Müller-Olm, M., Touili, T.: Regular Symbolic Analysis of Dynamic Networks of Pushdown Systems. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 473–487. Springer, Heidelberg (2005)
4. Bozzelli, L., Kretínský, M., Rehák, V., Strejcek, J.: On decidability of LTL model checking for process rewrite systems. Acta Inf. 46(1), 1–28 (2009)
5. Burkart, O., Caucal, D., Moller, F., Steffen, B.: Verification on infinite structures. In: Handbook of Process Algebra, pp. 545–623. Elsevier, North-Holland (2001)
6. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007)
7. de Moura, L., Bjørner, N.S.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
8. Esparza, J., Ganty, P.: Complexity of pattern-based verification for multithreaded programs. In: POPL, pp. 499–510 (2011)
9. Fisler, K.: Toward diagrammability and efficiency in event-sequence languages. Int. J. Softw. Tools Technol. Transf. 8(4), 431–447 (2006)
10. Ganty, P., Majumdar, R., Monmege, B.: Bounded underapproximations. Formal Methods in System Design 40(2), 206–231 (2012)
11. Göller, S., Lin, A.W.: Refining the Process Rewrite Systems Hierarchy via Ground Tree Rewrite Systems. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011 – Concurrency Theory. LNCS, vol. 6901, pp. 543–558. Springer, Heidelberg (2011)
12. Kozen, D.C.: Automata and Computability. Springer (2007)
13. Křetínský, M., Řehák, V., Strejček, J.: Extended Process Rewrite Systems: Expressiveness and Reachability. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 355–370. Springer, Heidelberg (2004)
14. Löding, C.: Infinite Graphs Generated by Tree Rewriting. PhD thesis, RWTH Aachen (2003)
15. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: POPL, pp. 283–294 (2011)
16. Maidl, M.: The common fragment of CTL and LTL. In: FOCS, pp. 643–652 (2000)
17. Mayr, R.: Decidability and Complexity of Model Checking Problems for Infinite-State Systems. PhD thesis, TU-Munich (1998)
18. Qadeer, S., Rehof, J.: Context-Bounded Model Checking of Concurrent Software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
19. Ramalingam, G.: Context-sensitive synchronization-sensitive analysis is undecidable. ACM Trans. Program. Lang. Syst. 22(2), 416–430 (2000)
20. Scarpellini, B.: Complexity of subcases of presburger arithmetic. Trans. of AMS 284(1), 203–218 (1984)

21. To, A.W.: Model Checking Infinite-State Systems: Generic and Specific Approaches. PhD thesis, LFCS, School of Informatics, University of Edinburgh (2010)
22. Torre, S.L., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS, pp. 161–170 (2007)
23. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS, pp. 332–344 (1986)
24. Verma, K.N., Seidl, H., Schwentick, T.: On the Complexity of Equational Horn Clauses. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 337–352. Springer, Heidelberg (2005)

Descriptional Complexity of Deterministic Regular Expressions

Katja Losemann[1,⋆], Wim Martens[1], and Matthias Niewerth[2,⋆⋆]

[1] Universität Bayreuth
[2] Technische Universität Dortmund

Abstract. We study the descriptional complexity of regular languages that are definable by deterministic regular expressions. First, we examine possible blow-ups when translating between regular expressions, deterministic regular expressions, and deterministic automata. Then we give an overview of the closure properties of these languages under various language-theoretic operations and we study the descriptional complexity of applying these operations. Our main technical result is a general property that implies that the blow-up when translating a DFA to an equivalent deterministic expression can be exponential.

1 Introduction

Deterministic or *one-unambiguous* regular expressions have been a topic of research since they were formally defined by Brüggemann-Klein and Wood in order to investigate a requirement in the ISO standard for the Standard Generalized Markup Language (SGML), where they were introduced to ensure efficient parsing. Today, the prevalent schema languages for XML data, such as Document Type Definition (DTD) and XML Schema, require that the regular expressions in their specification be deterministic. From a more foundational point of view, one-unambiguity is a natural manner in which to define determinism in regular expressions. As such, several decision problems behave better for deterministic regular expressions than for general ones. For example, language inclusion for regular expressions is PSPACE-complete but is tractable when the expressions are deterministic.

Although deterministic regular expressions are rather widespread and have been around for quite some time, they are not yet well-understood. This motivates us to study various foundational properties. In particular, we investigate the differences in the descriptional complexity between regular expressions (REs), deterministic regular expressions (DREs), and deterministic finite automata (DFA). Our initial motivation for this work was an unproved claim in

⋆ Supported by grant number MA 4938/21 of the Deutsche Forschungsgemeinschaft (Emmy Noether Nachwuchsgruppe).

⋆⋆ Supported by the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 643–654, 2012.

[2] which states that, for expressions of the form $\Sigma^* w$, where w is a Σ-string, every equivalent DRE is at least exponential in w. However, to the best of our knowledge, no proof for this result exists in the literature and proving it turned out to be rather non-trivial. Since this language has a polynomial-size RE and DFA, we needed to develop new techniques for proving lower bounds on the size of DREs.

A second set of contributions in this paper is a study of the effect of language-theoretic operations on languages that are definable by a DRE. In particular, we consider union, intersection, difference, concatenation, star, and reversal, for unary and arbitrary alphabets. We provide a complete overview of the closure properties of DRE-definable languages under these operations and we study the descriptional complexity of applying such operations on DREs and their DFAs. Several of these operations are relevant in XML schema management [7,17].

Until now, research on descriptional complexity of regular languages focused mainly on REs and DFAs. It is well-known that an exponential blow-up cannot be avoided when translating an RE into a DFA [12]. Ehrenfeucht and Zeiger [5] proved that there also exist DFAs which are exponentially more succinct than each equivalent RE. Gruber and Holzer [9,11] showed that there exist certain characteristics of automata which make equivalent regular expressions large. However, these characteristics cannot naïvely be transferred to DREs. For example, the languages used in the literature for proving lower bounds on the size of REs (e.g. [5,9,11]) are not definable by DREs.

The state complexity of boolean operations on DFAs is studied in [15,18,20], where in [18] the focus is on unary languages. In Section 4.2 we see that many results in [20] directly apply for DRE-definable languages, since they are on finite languages and every finite language is DRE-definable [1]. Gelade and Neven [8] and Gruber and Holzer [10] independently examined the descriptional complexity of complementation and intersection for REs. They showed that the size of the smallest RE for the intersection of a fixed number of REs can be exponential; and that the size of the smallest RE for the complement of an RE can be double-exponential. Furthermore, these bounds are tight. Gelade and Neven also investigate these operations on DREs and proved that the exponential bound on intersection is also tight when the input is given as DREs instead of REs [8]. Furthermore, they proved that the complement of a DRE can always be described by a polynomial-size RE. However, in their proofs, the languages of the resulting REs are not DRE-definable. Concatenation and reversal operations on regular languages are studied in [3,13,14,19,21], where in [21] also unary languages are examined.

2 Definitions

By Σ we always denote a finite alphabet of symbols. A (Σ-)*word* w over alphabet Σ is a finite sequence of symbols $a_1 \cdots a_n$, where $a_i \in \Sigma$ for each $i = 1, \ldots, n$. The set of all Σ-words is denoted by Σ^*. The *length* of a word $w = a_1 \cdots a_n$ is n and is denoted by $|w|$. The empty word is denoted by ε.

A *(deterministic, finite) automaton* (or *DFA*) A is a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, the transition function $\delta \subseteq Q \times \Sigma \to Q$ is a partial function, q_0 is the initial state and $F \subseteq Q$ is the set of accepting states. We sometimes abuse notation and denote a transition $\delta(q_1, a) = q_2$ by a tuple (q_1, a, q_2). We say that the aforementioned transition is q_1*-outgoing*, q_2*-incoming*, or a*-labeled*. The *run of A on word $w = a_1 \cdots a_n$* is a sequence $q_0 \cdots q_n$ where, for each $i = 1, \ldots, n$, $\delta(q_{i-1}, a_i) = q_i$. Word w is *accepted* by A if the run is *accepting*, i.e., if $q_n \in F$. By $L(A)$ we denote the *language of A*, i.e., the set of words accepted by A. By δ^* we denote the extension of δ to words, i.e., $\delta^*(q, w)$ is the state which is reached from q by reading w. In this paper we assume that all states of automata are *useful*, that is, every state can appear in some accepting run. This implies that, from each state in an automaton, an accepting state can be reached. The *size* $|A|$ of a DFA is the cardinality of $\{(q, a) \mid \delta(q, a)$ is defined$\}$.

The *regular expressions (RE)* over Σ are defined as follows: ε and every Σ-symbol is a regular expression; and whenever r and s are regular expressions then so are $(r \cdot s)$, $(r + s)$, and $(s)^*$. In addition, we allow $\emptyset$ as a regular expression, but we do not allow $\emptyset$ to occur in any other regular expression. We refer to Σ-symbols, ε, and $\emptyset$ as *atomic* expressions. For readability, we usually omit concatenation operators and parentheses in examples. The *language* defined by an RE r, denoted by $L(r)$, is defined as usual. Whenever we say that expressions or automata are *equivalent*, we mean that they define the same language. The *size* $|r|$ of r is defined to be the total number of occurrences of alphabet symbols, epsilons, and operators, i.e., the number of nodes in its parse tree. A regular expression r is *minimal* if there does not exist a regular expression r' with $L(r') = L(r)$ and $|r'| < |r|$. By *first(L)* we denote the set of all symbols $a \in \Sigma$, such that there is a word $aw \in L$. For a regular expression r, we define $\text{first}(r)$ as $\text{first}(L(r))$.

Deterministic regular expressions are defined as follows. Let $\bar{r}$ stand for the RE obtained from r by replacing, for every i and a, the i-th occurrence of alphabet symbol a in r (counting from left to right) by a_i. For example, for $r = b^*a(b^*a)^*$ we have $\bar{r} = b_1^*a_1(b_2^*a_2)^*$. A regular expression r is *deterministic* (or *one-unambiguous* [2] or a *DRE*) if there are no words wa_iv and wa_jv' in $L(\bar{r})$ such that $i \neq j$. The expression $(a + b)^*a$ is not deterministic since both strings a_2 and a_1a_2 are in $L((a_1 + b_1)^*a_2)$. The equivalent expression $b^*a(b^*a)^*$ is deterministic. Brüggemann-Klein and Wood showed that not every regular expression is equivalent to a deterministic one [2]. We call a regular language *DRE-definable* if there exists a DRE that defines it. The canonical example for a language that is not DRE-definable is $(a + b)^*a(a + b)$ [2].

3 Descriptional Complexity of DFAs, REs, and DREs

We consider the relative descriptional complexity of REs, DREs and DFAs. An overview of our results is shown in Figure 1. Since every DRE is an RE, we know that every minimal RE for a language L is smaller or equal to a minimal DRE

Finite Languages					Infinite Languages				
RE	DRE	DFA	Case exists?	Ref	RE	DRE	DFA	Case exists?	Ref
$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	yes	Obs. 1	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	yes	Obs. 1
$\Theta(n)$	$2^{\Omega(n)}$	$2^{\Omega(n)}$	yes	[15,2]	$\Theta(n)$	$2^{\Omega(n)}$	$2^{\Omega(n)}$	yes	Th. 6
$2^{\Omega(n)}$	$2^{\Omega(n)}$	$\Theta(n)$	no	[6]	$2^{\Omega(n)}$	$2^{\Omega(n)}$	$\Theta(n)$	?	
$\Theta(n)$	$2^{\Omega(n)}$	$\Theta(n)$	?		$\Theta(n)$	$2^{\Omega(n)}$	$\Theta(n)$	yes	Th. 15
$\Omega(n^{\log n})$	$\Omega(n^{\log n})$	$\Theta(n)$	yes	[11]					

Fig. 1. Overview descriptional complexity

for L. Furthermore, Brüggemann-Klein and Wood showed that, given a DRE r, one can construct a DFA A for $L(r)$ with size $O(|\Sigma||r|)$. Thus the table contains all substantial cases that ought to be considered.

We start with a trivial observation that shows that there are languages that do not cause any significant blow-up between the different representations. For example, consider the singleton $\{a^n\}$ and the infinite language $\{a^k \mid k \equiv 0 \mod n\} = L((aa \cdots a)^*)$ in which the latter expression has n occurrences of a.

Observation 1. There exists a class of finite languages $(L_n)_{n\in\mathbb{N}}$ and a class of infinite languages $(L'_n)_{n\in\mathbb{N}}$ such that, for each $n \in \mathbb{N}$, the minimal DFAs, minimal REs, and minimal DREs for L_n and L'_n have size $\Theta(n)$.

3.1 Finite Languages

We present an overview of what is known in the case of finite languages. For the language $(0+1)^{\leq n}1(0+1)^n$, Kintala and Wotschke, and Brüggemann-Klein and Wood showed that every DFA and every DRE has size exponential in n.

Theorem 2 ([15,2]). *For each $n \in \mathbb{N}$, the minimal DFA (and therefore every minimal DRE) for the language $(a+b)^{\leq n}a(a+b)^n$ have size $2^{\Omega(n)}$.*

Ellul et al. [6] showed that, for each DFA (or even non-deterministic automaton) A of size n that defines a finite language $L(A)$, there exists an RE for $L(A)$ of size $O(n^{\log n})$. Gruber and Johannsen showed that this upper bound is also tight. However, this problem was open for quite some time [11].

Theorem 3 ([6]). *Let A be a DFA of size n and let $L(A)$ be finite. Then there exists an RE r for $L(A)$ such that $|r| \leq O(n^{\log n})$.*

Theorem 4 ([11]). *There exists a family of finite languages $(L_n)_{n\in\mathbb{N}}$, such that the minimal DFA for L_n has $\Theta(n)$ states but every minimal RE for L_n has size $\Theta(n^{\log n})$.*

It remains open whether there exists a class of finite languages $(L_n)_{n\in\mathbb{N}}$, such that the minimal REs and the minimal DFA for L_n are exponentially more succinct than a minimal DRE for L_n.

3.2 Infinite Languages

In the case of infinite languages, it is well known that an exponential blow-up can occur when translating between REs and DFAs:

Theorem 5 ([12,5])
- *The minimal DFA for $(a+b)^*a(a+b)^n$ has size $2^{\Theta(n)}$.*
- *There exists a family of infinite regular languages $(L_n)_{n\in\mathbb{N}}$, s.t. the minimal DFA for L_n has size $\Theta(n^2)$ and every minimal RE for L_n has size $2^{\Omega(n)}$.*

However, to the best of our knowledge, all languages that are used in the literature to prove those blow-ups are not DRE-definable. Here, we prove that those blow-ups cannot be avoided for DRE-definable languages, too. For an exponential blow-up when translating an RE for a DRE-definable language to a DFA, we can extend the language of Theorem 2 to an infinite language.

Theorem 6. *For each $n \in \mathbb{N}$, the minimal DFA and every minimal DRE for the DRE-definable language $(a+b)^{\leq n}a(a+b)^n\#^*$ have size $2^{\Omega(n)}$.*

Next, we prove that there can be an exponential blow-up when translating a DFA to a DRE. The main idea of the proof is to identify concatenations of a minimal DRE in a DFA. Therefore, we search for *bottleneck states*, which are states through which every accepting run needs to go.

Definition 7. Let $A = (Q, \Sigma, \delta, q_0, Q_f)$ be a DFA. A state $q \in Q \setminus \{q_0\}$ is a *bottleneck state* of A if
- for every $w \in L(A)$ there are $v, z \in \Sigma^*$, s.t. $w = v \cdot z$ and $\delta^*(q_0, v) = q$, and
- if $q \in Q_f$, then $Q_f = \{q\}$ and there are $a \in \Sigma$ and $p \in Q$ s.t. $\delta(q, a) = p$.

Notice that we explicitly define initial states not to be bottleneck states.

Lemma 8. *Let $A = (Q, \Sigma, \delta, q_0, Q_f)$ be a DFA with a bottleneck state q. Then A has no equivalent DRE that is atomic or of the form s^*.*

In the following we show that accepting bottleneck states in a DFA identify concatenations in an equivalent minimal DRE. Therefore, let $A = (Q, \Sigma, \delta, q_0, Q_f)$ be a DFA. Then an equivalent DRE r is a *q-concatenation* if and only if $r = r_1 \cdot r_2$ and for every $v \in L(r_1)$ it holds that $\delta^*(q_0, v) = q$ in A. If r is a DRE with these conditions such that $L(r) \subsetneq L(A)$, then r is a *partial q-concatenation for A*.

Lemma 9. *Let $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ be a DFA for a DRE-definable language L, such that q_f is a bottleneck state of A. Then every minimal DRE r for L is a q_f-concatenation $r_1 \cdot r_2$ with $\mathrm{first}(r_2) = \{a \in \Sigma \mid \delta(q_f, a)$ is defined$\}$.*

Proof. By Lemma 8 it holds that r is neither atomic nor an expression s^*. It remains to show that r is neither a disjunction nor a concatenation which is not a q_f-concatenation. We can prove the following claim:

Claim 10. Let $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ be a DFA for a DRE-definable language L, such that q_f is a bottleneck state of A. Let $\emptyset \neq S \subseteq \mathrm{first}(L)$ and $r = r_1 r_2 \cdots r_n$ (with $n > 1$) be a minimal DRE for $L \cap S\Sigma^*$, such that no r_i is a concatenation. Then there exists an $i \in \{1, \ldots, n-1\}$ such that,

- *for every word* $w \in L(r_1 \cdots r_i)$, *it holds that* $\delta^*(q_0, w) = q_f$, *and*
- $\text{first}(r_{i+1} \cdots r_n) = \{a \in \Sigma \mid \delta(q_f, a)$ *is defined*$\}$.

In particular, this means that r *is a partial* q_f*-concatenation for* A.

We show why Claim 10 implies Lemma 9. From the discussion above, we know that r is either a concatenation or a disjunction. In the case that r is a concatenation, Claim 10 clearly implies the lemma (if $S = \text{first}(r)$, then $L \cap S\Sigma^* = L$).

We now show that, if r is a disjunction $(s_1 + \cdots + s_k)$, then r is not minimal, which contradicts the assumption we made about r. As an intermediate step we want to apply Claim 10 to every s_i. We therefore have to show that, for every i, (a) $L(s_i) = L \cap S_i\Sigma^*$ with $\emptyset \subsetneq S_i \subseteq \text{first}(L)$ and (b) s_i is a concatenation.

Since r is a DRE, it holds that $\text{first}(s_i) \cap \text{first}(s_j) = \emptyset$ for all $i \neq j$. Furthermore, we know that $\varepsilon \notin L$ and therefore $\varepsilon \notin L(s_i)$ for every i. Thus we can conclude that $L(s_i) = L \cap S_i\Sigma^*$ with $S_i = \text{first}(s_i) \subseteq \text{first}(r)$ for every i. This proves (a). Notice that $S_i \neq \emptyset$ because r is minimal. Next we prove that every s_i is a concatenation. W.l.o.g., s_i is not a disjunction. Since $\varepsilon \notin L(s_i)$, s_i is not of the form t^*. Now take an arbitrary $a \in S_i$. Then there exists a word $aw \in L(r)$ with $w \neq \varepsilon$, because q_f has at least one outgoing transition. Since r is a DRE, $L(s_i)$ contains all words $b \cdot v \in L$ where $b \in S_i$ and $v \in \Sigma^*$, and therefore $aw \in L(s_i)$. As $|aw| > 1$, s_i cannot be atomic. The only remaining possibility is that s_i is a concatenation, which proves (b). Also, s_i is a minimal DRE.

We can now apply Claim 10 to every s_i and conclude that we can write every s_i as $s_i^a s_i^b$ such that (i) $\delta(q_0, w) = q_f$ for every $w \in L(s_i^a)$ and (ii) $\text{first}(s_i^b) = \{a \in \Sigma \mid \delta(q_f, a) \text{ is defined}\}$. Notice that s_i^a and s_i^b can be concatenations again.

Let $A^{q_f} = (\Sigma, Q, \delta, q_f, \{q_f\})$ be the automaton A where the initial state is q_f. From (i) and (ii), we can conclude that $L(s_i^b) = L(A^{q_f})$ for every i. Therefore all expressions s_i^b are equivalent. Thus, r can equivalently be written as $(s_1^a + \cdots + s_k^a)s_1^b$, which is strictly smaller than r. This contradicts the minimality of r and therefore contradicts that r is a disjunction, which concludes the proof. □

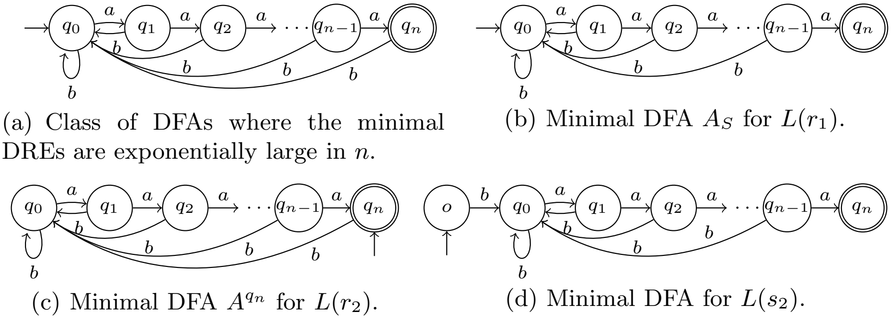
Notice that a DRE can have multiple q_f-concatenations. For example, the expression $a \cdot b^* \cdot (c \cdot b^*)^*$ has a DFA with a unique accepting state q_f and has two q_f-concatenations. However, a DRE can only have one q_f-concatenation of the form $r_1 \cdot r_2$ where $\text{first}(r_2) = \{a \mid \delta(q_f, a) \text{ is defined}\}$. Furthermore, if $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ is a DFA with a bottleneck state q_f, it holds that $L(A)$ is infinite. Lemma 9 gives us a rather precise structure of each minimal DRE $r_1 \cdot r_2$. The following lemma also clarifies $L(r_1)$ and $L(r_2)$.

Lemma 11. *For a DFA* $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ *with a bottleneck state* q_f *let the* q_f*-concatenation* $r_1 \cdot r_2$ *be an equivalent minimal DRE with* $\text{first}(r_2) = \{a \in \Sigma \mid \delta(q_f, a)$ *is defined*$\}$. *Then*

(1) $L(r_1) = L(A_S)$ *where* $A_S = (Q, \Sigma, \delta - S, q_0, \{q_f\})$, $S = \{(q_f, a, q) \in \delta \mid a \in \Sigma, q \in Q\}$; *and*

(2) $L(r_2)$ *is infinite where* $L(r_2) = L(A^{q_f})$ *with* $A^{q_f} = (Q, \Sigma, \delta, q_f, \{q_f\})$.

Note that (2) follows from the proof of Lemma 9.



(a) Class of DFAs where the minimal DREs are exponentially large in n.

(b) Minimal DFA A_S for $L(r_1)$.

(c) Minimal DFA A^{q_n} for $L(r_2)$.

(d) Minimal DFA for $L(s_2)$.

Fig. 2. Minimal DFAs for subexpressions from the proof of Lemma 14

Before we can finally prove the blow-up from DFA to DRE, we need two general results on minimal DREs. The first is a very straightforward relation between a state and a concatenation in the DRE. Therefore, we say that a regular language L is *prefix-free* if and only if, for every word $v \in L$, there exists no $z \in \Sigma^*$ such that $v \cdot z \in L$.

Lemma 12. *Let $L_a = L \cdot \{a\}$ be a prefix-free regular language. Then there exists a minimal DRE for L_a which is either a or of the form $r \cdot a$.*

Lemma 13. *If r^* is a minimal DRE, then $\varepsilon \notin L(r)$.*

However, the DRE r in a minimal DRE r^* in Lemma 13 can still contain ε-symbols. This holds, for example, for the DRE $r = (a(b+\varepsilon))^*$.

Now we are ready to prove the exponential blow-up when translating DFAs to DREs. In particular, we prove that every minimal DRE for the DFA in Figure 2(a) is exponential in n. We denote the language of this DFA with $L^{[n]}$.

Lemma 14. *There is a minimal DRE r for $L^{[n]}$ containing at least 2^n concatenations.*

Proof. Let A be the minimal DFA for $L^{[n]}$ (see Figure 2(a)). The proof is by induction on n. For the induction basis, let $n = 1$. Since A has an accepting bottleneck state, we know by Lemma 9 that r is a concatenation $r_1 \cdot r_2$ with $\text{first}(r_2) = \{b\}$. By Lemma 11, it follows that $L(r_1) = L(b^*a)$. This implies that there is a minimal DRE for $L^{[1]} = L(b^* \cdot a \cdot r_2)$, with at least 2 concatenations.

For the induction step, assume that there exists a minimal DRE for $L^{[n-1]}$ containing at least 2^{n-1} concatenations. By Lemma 9, r is a q_n-concatenation $r_1 \cdot r_2$ with $\text{first}(r_2) = \{b\}$. Lemma 11 implies that the automaton in Figure 2(b) is a DFA for $L(r_1)$ and the automaton in Figure 2(c) is a DFA for $L(r_2)$.

Next we show that r_1 and r_2 each contain a subexpression for the language $L^{[n-1]}$. For r_1 we observe that $L(r_1)$ (see Figure 2(b)) is prefix-free and a language of the form $L' \cdot \{a\}$. Thus there exists a minimal DRE r_1 of the form $s_1 \cdot a$ by Lemma 12, where $L(s_1)$ is defined by the DFA of Figure 2(b) without the transition $\delta(q_{n-1}, a) = q_n$ and with q_{n-1} as accepting state. As we can see, this is a DFA for $L^{[n-1]}$; hence $L(s_1) = L^{[n-1]}$. Then, by induction hypothesis there exists a DRE s_1, such that s_1 and therefore r_1 contain at least 2^{n-1} concatenations.

For r_2, we observe that $L(r_2)$ is infinite (see A^{q_n} in Figure 2(c)), which implies that r_2 is not an atomic expression. Furthermore, it holds that $|\,\mathrm{first}(r_2)| = 1$ and $\varepsilon \in L(r_2)$. It follows that r_2 cannot be a concatenation $r_2 = r_3 \cdot r_4$, as the first sets of r_3 and r_4 would have to be disjunct because of $\varepsilon \in r_3$. Next we show that r_2 cannot be a disjunction. Since $\mathrm{first}(r_2) = \{b\}$, the only possible disjunction is $r_2 = b \cdot r_3 + \varepsilon$ for some DRE r_3. As $\delta(q_n, b) = q_0$ in A^{q_n}, we observe that $L(r_3) = L^{[n]}$, which directly contradicts that r is a minimal DRE for $L^{[n]}$.

Thus r_2 is an expression of the form s_2^*. Next we investigate the structure of a DFA for $L(s_2)$. For every word $v \in L(s_2)$, it holds that $\delta^*(q_n, v) = q_n$ in A^{q_n}. Since s_2^* is a DRE and $\mathrm{first}(r_2) = \{b\}$, $L(s_2)$ cannot contain a word v such that $v = w \cdot z$ with $w, z \neq \varepsilon$ and $\delta^*(q_n, w) = q_n$. These properties characterize $L(s_2)$, for which the minimal DFA is shown in Figure 2(d). Because the DFA has a bottleneck state q_1, s_2 cannot be atomic or an expression t^* by Lemma 8. Furthermore, s_2 is not a disjunction, because $|\,\mathrm{first}(s_2)| = 1$, $\varepsilon \notin L(s_2)$, and s_2 is a DRE. Thus s_2 is a concatenation $b \cdot t$, where $L(t)$ is definded by the DFA from Figure 2(d) without the transition (o, b, q_0) and with q_0 as initial state. By Lemma 12, it follows that $s_2 = b \cdot t \cdot a$, where $L(t) = L^{[n-1]}$. Thus, by induction hypothesis, t and therefore r_2 contain at least 2^{n-1} concatenations.

Finally, it holds that r_1 and r_2 contain at least 2^{n-1} concatenations each, i.e., $r = r_1 \cdot r_2$ contains at least 2^n concatenations. This concludes the proof. □

Since we can write $L^{[n]} = L((b + ab + \cdots + a^n b)^* a^n) = L((b(a + b(\cdots(ab + b)\cdots)))^* a^n)$, we obtain the following theorem:

Theorem 15. *For each $n \in \mathbb{N}$, the minimal DFA for $L^{[n]}$ has size $\Theta(n)$, every minimal RE for $L^{[n]}$ has size $\Theta(n)$, and every minimal DRE has size $2^{\Omega(n)}$.*

3.3 Application on an Example from the Literature

Brüggemann-Klein and Wood claimed that every minimal DRE for languages of the form $\Sigma^* a_1 \cdots a_n$, where $a_1 \cdots a_n$ is a fixed Σ-word, is exponential [2]. However, to the best of our knowledge, no proof for this result exists in the literature. We prove this claim by using bottleneck states. Therefore we will generalize the special structure of the automata of languages $L^{[n]}$ (see Figure 2(a)) and $L(\Sigma^* a_1 \cdots a_n)$ to provide a formal proof.

Definition 16. Let $A = (Q, \Sigma, \delta, o, \{q_n\})$ be a DFA with $\Sigma \supseteq \{a_1, \ldots, a_n\}$ and $Q \supseteq \{q_0, \ldots, q_n\}$. Then A contains a *bottleneck tail* of length n, if all of the following hold:

1. q_i is a bottleneck state for every $i \in \{0, \ldots, n\}$;
2. $(q_{i-1}, a_i, q_i) \in \delta$ for all $i \in \{1, \ldots, n\}$;
3. for every $i \in \{0, \ldots, n\}$ there is an $a \in \Sigma$ and a transition (q_i, a, o) in A; and
4. for every $i \in \{1, \ldots, n\}$, if $(q, a, q_i) \in \delta$ then $q = q_{i-1}$ and $a = a_i$.

Op.	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert \geq 1$	Op.	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert \geq 1$	Op.	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert \geq 1$
$\setminus$	no	no	$\cup$	no	no	$\cdot$	no	no
Rev	yes	no	$\cap$	yes	no	*	yes	no

Fig. 3. Closure Properties of DRE-definable languages

For example, the automaton in Figure 2(a) and the minimal DFA for $L(\Sigma^* a_1 \cdots a_n)$ each contain a bottleneck tail of length $n - 1$. We prove that a bottleneck tail causes a blow-up in a DRE, exponential in the length of the tail.

Theorem 17. *Let $A = (Q, \Sigma, \delta, o, \{q_n\})$ be a DFA for a DRE-definable regular language L with a bottleneck tail of length n. Then there exists a minimal DRE r for L which contains at least 2^n concatenations.*

Theorem 18. *Every minimal DRE for $L(\Sigma^* a_1 \cdots a_n)$ has size $2^{\Omega(n)}$.*

4 Operations on DRE-Definable Languages

We investigate the descriptional complexity of several language-theoretic operations on DREs and their DFAs. Most results concern DFAs for DRE-definable languages, which allows us to infer lower bounds for DREs as well. First, we present an overview of the closure properties of DRE-definable languages.

4.1 Closure Properties of DRE-Definable Languages

It has been observed that DRE-definable languages are not closed under union [2], intersection [16,4] or complement [8]. DRE-definable languages are also not closed under concatenation [2], reversal[1] (take $L((a + b)^* a(a + b))$) or Kleene star [2]. These results hold for alphabets with at least two symbols. For unary alphabets, the same results hold, except for reversal, intersection and star. In these three cases, we prove that DRE-definable languages are closed. It is easy to see that DRE-definable languages over unary alphabets are closed under reversal, since for unary alphabets the language and its reversal are equal. The other two cases are non-trivial. The results are summarized in Figure 3.

Theorem 19. *DRE-definable regular languages over a unary alphabet are closed under reversal, intersection, and Kleene star.*

4.2 Descriptional Complexity of Operations on DRE-Definable Languages

We are now ready to apply previously obtained results to prove lower bounds on the descriptional complexity of operations on DREs. From Section 4.1 we know that we need to be careful that the language after performing the operations is indeed DRE-definable. We first prove some lower bounds directly on DREs. For DRE-definable languages we get the following by Theorem 2 and 15.

[1] The reversal of a language L is the set of strings $\{a_n \cdots a_1 \mid a_1 \cdots a_n \in L\}$.

Theorem 20. *There exist regular languages $(L_n)_{n\in\mathbb{N}}$ such that, for each $n \in \mathbb{N}$, the minimal DREs for L_n have size $\Theta(n)$, whereas the minimal DREs for the reversal of L_n have size $2^{\Theta(n)}$. This holds in the case where all L_n are finite languages and in the case where all L_n are infinite languages.*

Indeed, in the finite case one could take L_n to be $L((a+b)^{\leq n}a(a+b)^n)$ and in the infinite case take the language $L^{[n]}$ from Theorem 15.

Theorem 21. *There exist regular languages $(L_n^1)_{n\in\mathbb{N}}$ and $(L_n^2)_{n\in\mathbb{N}}$ such that, for each $n \in \mathbb{N}$, the minimal DREs for L_n^1 and L_n^2 have size $\Theta(n)$ and the minimal DREs for $L_n^1 \cdot L_n^2$ have size $2^{\Theta(n)}$. This holds in the case where all L_n^1 and L_n^2 are finite languages and in the case where all L_n^1 and L_n^2 are infinite languages.*

Indeed, in the finite case we can take $L_n^1 = (a+b)^{\leq n}$ and $L_n^2 = a(a+b)^n$ and in the infinite case we can take $L_n^1 = (a+b)^*$ and $L_n^2 = a^n cc^*$.

The following results do not immediately concern the minimal size of DREs after performing an operation, but focus on the minimal size of the DFAs for the DREs. For DRE-definable languages, lower bounds can always be transfered. In some cases, we can even infer upper bounds on the DRE size. Consider, for example, the case of languages over a unary alphabet: For those languages all minimal DREs have size linear in the minimal DFA.

Theorem 22. *Let A be a minimal DFA with m states for a DRE-definable language L over a unary alphabet. Then there exists a minimal DRE r for L, such that r is of size $O(m)$.*

To this end, for a DRE-definable language L, we write DDFA(L) for the minimal DFA defining L. We summarize our results in Figure 4 and 5, where in each case we consider a single use of a boolean operation and a k-times application. In Figure 5 the resulting DFA has to define an infinite language.

It is well-known that for the complement on DFAs there is no blow-up [12]. Since all finite languages are DRE-definable, we provide the known results of Yu [20] separated in Figure 4. For all remaining operations the upper bounds are obtained by the standard product construction [12]. For the union and intersection of two finite languages an exact result is as far as we know still open.

Theorem 23. *For every $k \in \mathbb{N}$ there exists finite languages $L_1, \ldots, L_k$, such that the minimal DFA for every L_i has $\Theta(k)$ states and the minimal DFA for $L_1 \cap \cdots \cap L_k$ or $L_1 \cup \cdots \cup L_k$ has at least $2^{\Theta(k)}$ states.*

The theorem is obtained by taking $L_i = \{x_1 \ldots x_k y_k \ldots y_1 \in \{a,b\}^* \mid x_i = y_i\}$.

Now we examine DDFAs which are the result of a boolean operation on $k \geq 2$ infinite DRE-definable languages. For general DFAs the descriptional complexity is studied in [18,20]. In the following we show that for infinite DRE-definable languages the complexity remains the same in almost all cases. Only for the union of two DDFAs the descriptional complexity is strictly lower than for DFAs.

	$\|\Sigma\| = 1$		$\|\Sigma\| \geq 1$	
	1	k	1	k
$\setminus$	$\Theta(m)$ [12]	—	$\Theta(m)$ [12]	—
$\cap$	$\Theta(\min\{m_1, m_2\})$ [20]	$\Theta(\min\{m_1, ..., m_k\})$ [20]	$O(m_1 m_2)$ [20]	$2^{\Omega(k)}$ (Th. 23)
$\cup$	$\Theta(\max\{m_1, m_2\})$ [20]	$\Theta(\max\{m_1, ..., m_k\})$ [20]	$O(m_1 m_2)$ [20]	$2^{\Omega(k)}$ (Th. 23)

Fig. 4. Descriptional complexity of minimal DFAs for finite languages

	$\|\Sigma\| = 1$		$\|\Sigma\| \geq 1$	
	1	k	1	k
$\setminus$	$\Theta(m)$ [12]	—	$\Theta(m)$ [12]	—
$\cap$	$\Theta(m_1 m_2)$ (Th. 24)	$2^{\Omega(k)}$ (Th. 24)	$\Theta(m_1 m_2)$ (Th. 24)	$2^{\Omega(k)}$ (Th. 24)
$\cup$	$\Theta(\max\{m_1 m_2\})$ (Th. 25)	$\Theta(\max\{m_1, ..., m_k\})$(Th. 25)	$O(m_1 m_2)$	$2^{\Omega(k)}$ (Th. 26)

Fig. 5. Descriptional complexity of minimal DFAs for inifinite DRE-definable languages

Theorem 24. *For each $k \in \mathbb{N}$, there exist infinite DRE-definable languages $L_1, \ldots, L_k$, such that, for every $i \in \{1, \ldots, k\}$ the minimal DFA for L_i has $O(k \log k)$ states and DDFA$(L_1 \cap \cdots \cap L_k)$ has $k^{\Omega(k)}$ states. This holds even when the alphabet is unary.*

The theorem is obtained by k languages $L_i = (a^{m_i})^*$ with $1 \leq i \leq k$ and k different m_i, such that $\gcd(m_i, m_j) = 1$ for each pair (m_i, m_j).

At last we examine the union of DFAs for DRE-definable languages where the result still describes a DRE-definable language. We get that for DFAs over unary alphabets the complexity is only linear; hence is strictly lower than for intersection. For arbitrary alphabets the complexity is again exponential.

Theorem 25. *Let $L_1, \ldots, L_k$ be infinite languages over a unary alphabet, such that the minimal DFAs for every L_i with $i \in \{1, \ldots, k\}$ has m_i states. Then the DDFA A for $L_1 \cup \cdots \cup L_k$ has $\Theta(\max\{m_1, \ldots, m_k\})$ states.*

Theorem 26. *For each $k \in \mathbb{N}$, there are infinite DRE-definable languages L_1, $\ldots, L_k$ such that, for each $i \in \{1, \ldots, k\}$ there is a DFA of size $\Theta(k)$ for L_i, but DDFA$(L_1 \cup \cdots \cup L_k)$ has size $2^{\Theta(k)}$.*

The theorem follows from taking $L_i = \{x_1 \ldots x_k y_k \ldots y_1 w \in \{a, b\}^* \mid x_i = y_i\}$.

5 Conclusions and Further Work

In this paper we were motivated by the aim to come to a better understanding of DRE-definable languages. For example, we developed a new technique to prove lower bounds on the size of DREs by using bottleneck states and tails in a DFA. As a consequence of this technique, we now know that, when translating an RE into a DFA and when translating a DFA into a DRE, an exponential blow-up cannot be avoided. However, we do not know yet whether there are DRE-definable languages for which a translation from an RE to a DRE causes a double exponential blow-up.

Finally we examine several operations on DRE-definable languages. We obtain an overview of the closure properties and the descriptional complexity of these operations on DRE-definable languages. A tight lower bound for the union of two DFAs for DRE-definable languages remains open.

References

1. Bex, G.J., Gelade, W., Martens, W., Neven, F.: Simplifying XML Schema: effortless handling of nondeterministic regular expressions. In: SIGMOD (2009)
2. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. Information and Computation 142(2), 182–206 (1998)
3. Câmpeanu, C., Culik, K., Salomaa, K., Yu, S.: State Complexity of Basic Operations on Finite Languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
4. Caron, P., Han, Y.-S., Mignot, L.: Generalized One-Unambiguity. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 129–140. Springer, Heidelberg (2011)
5. Ehrenfeucht, A., Zeiger, H.: Complexity measures for regular expressions. JCSS 12(2), 134–146 (1976)
6. Ellul, K., Krawetz, B., Shallit, J., Wang, M.: Regular expressions: new results and open problems. In: JALC, pp. 233–256 (2004)
7. Gelade, W., Idziaszek, T., Martens, W., Neven, F.: Simplifying XML Schema: Single-type approximations of regular tree languages. In: PODS (2010)
8. Gelade, W., Neven, F.: Succinctness of the complement and intersection of regular expressions. In: TOCL, pp. 4:1–4:19 (2012)
9. Gruber, H., Holzer, M.: Finite Automata, Digraph Connectivity, and Regular Expression Size. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 39–50. Springer, Heidelberg (2008)
10. Gruber, H., Holzer, M.: Tight Bounds on the Descriptional Complexity of Regular Expressions. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 276–287. Springer, Heidelberg (2009)
11. Gruber, H., Johannsen, J.: Optimal Lower Bounds on Regular Expression Size Using Communication Complexity. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 273–286. Springer, Heidelberg (2008)
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (2007)
13. Jirásek, J., Jirásková, G., Szabari, A.: State Complexity of Concatenation and Complementation of Regular Languages. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 178–189. Springer, Heidelberg (2005)
14. Jirásková, G.: On the State Complexity of Complements, Stars, and Reversals of Regular Languages. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 431–442. Springer, Heidelberg (2008)
15. Kintala, C., Wotschke, D.: Amounts of nondeterminism in finite automata. Acta Informatica 13, 199–204 (1980)
16. Losemann, K.: Boolesche Operationen auf deterministischen regulären Ausdrücken. Master's thesis, TU Dortmund (October 2010)
17. Martens, W., Niewerth, M., Schwentick, T.: Schema design for XML repositories: Complexity and tractability. In: PODS (2010)
18. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function. In: IJFCS, pp. 145–159 (2002)
19. Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages. In: TCS, pp. 315–329 (2004)
20. Yu, S.: State complexity of regular languages. In: JALC, p. 221 (2001)
21. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. In: TCS, pp. 315–328 (1994)

Identity Testing, Multilinearity Testing, and Monomials in Read-Once/Twice Formulas and Branching Programs*

Meena Mahajan[1], B.V. Raghavendra Rao[2], and Karteek Sreenivasaiah[1]

[1] Institute of Mathematical Sciences, Chennai, India
{meena,karteek}@imsc.res.in
[2] Saarland University, Saarbrücken, Germany
bvrr@cs.uni-sb.de

Abstract. We study the problem of testing if the polynomial computed by an arithmetic circuit is identically zero (ACIT). We give a deterministic polynomial time algorithm for this problem when the inputs are read-twice formulas. This algorithm also computes the MLIN predicate, testing if the input circuit computes a multilinear polynomial.

We further study two related computational problems on arithmetic circuits. Given an arithmetic circuit C, 1) ZMC: test if a given monomial in C has zero coefficient or not, and 2) MonCount: compute the number of monomials in C. These problems were introduced by Fournier, Malod and Mengel [STACS 2012], and shown to characterize various levels of the counting hierarchy (CH).

We address the above problems on read-restricted arithmetic circuits and branching programs. We prove several complexity characterizations for the above problems on these restricted classes of arithmetic circuits.

1 Introduction

A fundamental question concerning a given arithmetic circuit is: does the circuit compute the identically zero polynomial? This is the well-known problem Arithmetic Circuit Identity Testing ACIT, that has spurred an enormous amount of research in the last two decades. A complete derandomization of black-box ACIT even for depth four arithmetic circuits implies circuit lower bounds [13,2].

Today, there are two frontiers for identity testing. One is based on the (alternation) depth of the circuit. Deterministic identity testing algorithms are known for depth-2 crcuits, for depth-3 circuits with restrictions on the top fanin, and for restricted depth-4 circuits. (See [1] and the references therein.) As indicated by [2], improving this to arbitrary depth-4 circuits will be a major breakthrough.

The other frontier is concerned with formulas. Restricting fanout in a circuit to 1 yields formulas; further restricting formulas to allow each variable at no more than k leaves yields Read-k Formulas. The simplest kind of formulas are read-once formulas ROFs: every variable appears at most once. Deterministic

* Partially supported by Indo-German Max Planck Center (IMPECS).

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 655–667, 2012.

polynomial-time algorithms for ACIT on such formulas are trivial. Going beyond these for $k > 1$, one breakthrough in [15] shows how to test k-sums of ROFs: for each $k \in O(1)$, ACIT can be efficiently performed on a sum of k ROFs.

However, not every Read-k formula can be expressed as a sum of k ROFs. Along this thread, the next improvement in [5] shows how to do identity testing on read-k formulas that are known to be multilinear, that is, the polynomials computed at each node are multilinear.

To use the algorithm from [5] for a Read-k formula, we first need to check whether it is multilinear. The multilinearity testing predicate MLIN is as hard as ACIT in general ([11]), but for read-k formulas, it could conceivably be easier. Thus one way to extend the result of [5] to arbitrary read-k formulas is to develop a multilinearity test for such formulas.

Our main result is a multilinearity test for read-twice formulas R2Fs. Such a test, in conjunction with [5], would give an ACIT test for R2Fs too. But our test is actually intertwined with an ACIT test for subformulas. We give a deterministic polynomial-time algorithm that simultaneously decides whether an R2F is multilinear and whether it is identically zero. It performs identity tests on partial derivatives. It also uses the sum-of-k-ROFs test from [15] on some subformulas as well as on some formulas obtained by transforming subformulas of the input formula. Thus it is inherently a non-blackbox algorithm; so is the polynomial-time algorithm from [15].

ACIT tests check whether the polynomial computed by the crcuit has at least one monomial. Natural generalizations/variants of this question are (1) MonCount: compute the number of monomials in the polynomial computed by a given circuit, and (2) ZMC: Decide whether a given monomial has zero coefficient in the polynomial computed by a given circuit. ZMC was introduced by Koiran and Perifel [14]. More recently, Fournier, Malod and Mengel [11] showed that ZMC and MonCount characterize certain levels of the counting hierarchy (CH, the hierarchy based on the complexity classes PP and $\mathsf{C_{=}P}$). In fact, MonCount remains hard even if restricted to formulas. They also show that if the circuits compute multilinear polynomials, then these problems become easier (equivalent to PP and ACIT respectively), and that multilinearity checking itself is equivalent to ACIT. All these results from [11] are in the non-black-box model, where the circuit is given explicitly in the input.

Since ACIT on Read-k formulas appears easier, naturally one could ask whether MonCount and ZMC become easier as well? We observe that this is not the case: even for monotone (no negative constants) read-twice formulas, MonCount is #P-hard. This further leads us to the investigation: where exactly does hardness for MonCount and ZMC begin? Further, translating the classes between NP and PSPACE down to classes below P, can we show that on restricted circuits, MonCount and ZMC are complete for the translated classes?

Starting with ROFs, we show (Theorem 2) that MonCount for ROFs is in the GapNC^1-hierarchy, i.e. the AC^0-closure of GapNC^1, where GapNC^1 is the class of Boolean problems that can be computed by arithmetic formulas over the integers with constants 0, 1, -1. The GapNC^1-hierarchy is an intriguing class that lies between NC^1 and DLOG and has been studied extensively in the last two decades; see for instance [3]. We also show that ZMC for ROFs is in logspace (Theorem 6). It is straightforward to see that ZMC for ROFs is hard for $\mathsf{C_=NC}^1$, so this is almost tight. (The "gap" between Boolean NC^1, $\mathsf{C_=NC}^1$, GapNC^1 and DLOG is very small.)

Another natural, well-studied restriction is when the circuit is an algebraic branching program BP with edges labeled by the allowed constants or by variables. Evaluation of BPs on Boolean-valued inputs is complete for the arithmetic class GapL, the logspace analogue of the class GapP. The GapL hierarchy (the AC^0 closure of GapL) is known to be contained in $\log n$ depth threshold circuits TC^1 and hence in $\log^2 n$ depth Boolean circuits NC^2. Two restrictions on BPs, in order of increasing generality, are: (1) occur-once BPs, or OBPs, where each variable appears at most once anywhere in the BP, these subsume ROFs, and (2) multilinear BPs, or MBPs, where the polynomial computed at every node is multilinear. Again, deterministic algorithms are known for ACIT on OBPs, [12]. We show that $\mathsf{MonCount}$ for OBPs is in the GapL hierarchy (Theorem 4), while ZMC for OBPs and even MBPs is complete for the complexity class $\mathsf{C_=L}$ (Theorem 5). (As a comparison, a well-known complete problem for $\mathsf{C_=L}$ is testing singularity of an integer matrix [4].)

A related problem explored in [11] as a tool to solving $\mathsf{MonCount}$ is that of checking, given a circuit C and monomial m, whether C computes any monomial that extends m. Denote this problem $\mathsf{ExistExtMon}$. Though our algorithms for $\mathsf{MonCount}$ do not need this subroutine, we also show that for OBPs (and hence for ROFs), $\mathsf{ExistExtMon}$ lies in the GapL hierarchy (Theorem 7).

2 Preliminaries

Circuits, Formulas, Branching Programs, Polynomials. An *arithmetic circuit* C over a ring R is a directed acyclic graph where every node has in-degree zero or two. The nodes with in-degree zero are called leaves. Internal nodes are labeled $+$ or $\times$ and leaves are labeled from $X \cup R$, where $X = \{x_1, \ldots, x_n\}$, a set of variables. There is a node of out-degree zero, called the root node or the output gate. Unless otherwise stated, R is the ring of integers $\mathbb{Z}$, and we allow only the constants $\{-1, 0, 1\}$ in the circuits. An *arithmetic formula* F is an arithmetic circuit where fan-out for every gate is at most one.

The depth of a circuit is the length of a longest root-to-leaf path. The alternation-depth is the maximum number of alternations between $+$ and $\times$ gates along any root-to-leaf path. In the literature on identity testing, depth is used to mean alternation-depth. However we differentiate between these, as is done in uniform circut complexity literature, because bounded fanin is crucial to some of our algorithms. Note that converting a circuit to a bounded fanin circuit increases only the depth, not the size or the alternation depth.

Every node in C computes a polynomial in $R[x_1, \ldots, x_n]$ in a natural way. Let g be a gate in a circuit (or formula) C. We denote by p_g the polynomial computed at gate g of C. We denote by p_C the polynomial p_r, where r is the output gate of C. Let $\mathsf{var}_g \overset{\Delta}{=} \{x_i \mid \text{some descendant of } g \text{ is a leaf labelled } x_i\}$.

A *read-once* arithmetic formula (ROF for short) is an arithmetic formula where each variable occurs at most once as a label. More generally, in a *read-k* arithmetic formula a variable occurs at most k times as a label.

An algebraic branching program (ABP for short) over a ring R is a directed acyclic graph B with edges labeled from $\{x_1, \ldots, x_n\} \cup R$, and with two designated nodes, s with zero in-degree, and t with zero out-degree. For any directed path ρ in B, let $\mathsf{weight}(\rho) = \prod_{e:\text{ an edge in }\rho} \mathsf{label}(e)$.

Any pair of nodes u, v in B computes a polynomial in $R[x_1 \ldots x_n]$ defined by: $p_B(u, v) = \sum_{\rho:\ \rho\text{ is a }u \rightsquigarrow v\text{ path in }B} \mathsf{weight}(\rho)$. The ABP B computes the polynomial $p_B \stackrel{\triangle}{=} p_B(s, t)$. We drop the subscript B when clear from context.

We consider the following restrictions of ABPs in increasing order of generality: (1) occur-once ABPs (OBP for short), where each variable appears at most once anywhere in the ABP (such BPs generalize ROFs), (2) read-once ABPs, or RBPs, where no path has two occurrences of the same variable, and (3) multilinear BPs, or MBPs, where the polynomial computed at every pair of nodes is multilinear.

Complexity Classes. For all the standard complexity classes, the reader is referred to [6]. We define some of the non-standard complexity classes that are used in the paper. Let $f = (f_n)_{n \geq 0}$ be a family of integer valued functions $f_n : \{0, 1\}^n \to \mathbb{Z}$. f is in the complexity class GapL exactly when there is some nondeterministic logspace machine M such that for every x, $f(x)$ equals the number of accepting paths of M on x minus the the number of rejecting paths of M on x. $\mathsf{C_=L}$ is the class of languages L such that for some $f \in \mathsf{GapL}$, for every x, $x \in L \Leftrightarrow f(x) = 0$. The GapL hierarchy is built over bit access to GapL functions, with a deterministic logspace machine at the base, and is known to be contained in NC^2. (See [4,3] for more details.)

GapNC^1 denotes the class of families of functions $(f_n)_{(n \geq 0)}$, $f_n : \{0, 1\}^n \to \mathbb{Z}$, where $(f_n)_{n>0}$ can be computed by a uniform polynomial size log depth arithmetic circuit family. This equals the class of functions computed by uniform polynomial-sized arithmetic formulas ([8]). $\mathsf{C_=NC}^1$ is the class of languages L such that for some GapNC^1 function family $(f_n)_{n \geq 0}$, and for every x, $x \in L \Leftrightarrow f_{|x|}(x) = 0$. The GapNC^1 hierarchy comprises of languages accepted by polynomial-size constant depth unbounded fanin circuits (AC^0) with oracle access to bits of GapNC^1 functions, and is known to be contained in DLOG. (See [9,10] for more details.)

Miscellaneous Notation. A monomial is represented by the sequence of degrees of the variables. For instance, over x_1, x_2, x_3, the monomial x_1^2 is represented as $(2, 0, 0)$, and the monomial $x_1 x_3$ is represented as $(1, 0, 1)$. For a degree sequence $m = (d_1, \ldots, d_n)$ we denote the monomial $\prod_{i=1}^n x_i^{d_i}$ by X^m. For any set $S \subseteq [n]$, we denote by m_S the multilinear monomial $\prod_{i \in S} x_i$. For a monomial m and polynomial p, $\mathsf{coeff}(p, m)$ denotes the coefficient of m in p. [statement S] is a Boolean 0-1 valued predicate that takes value 1 exactly when S is true.

We now describe the computational problems considered in this paper.

ACIT: Given an arithmetic circuit C over $\mathbb{Z}$, test if the polynomial computed by C is identically zero.

MonCount: Given an arithmetic circuit C over $\mathbb{Z}$, compute the number of monomials in the polynomial computed by C.

MLIN: Given an arithmetic formula F over $\mathbb{Z}$, test if the polynomial p_F is multilinear.

ZMC: Given an arithmetic circuit C over $\mathbb{Z}$, and a monomial m, test if $\mathsf{coeff}(p_C, m)$ is zero or not.

ExistExtMon: Given an arithmetic circuit C over Z, and a monomial m, test if there is a monomial M with non-zero coefficient in p_C such that M extends m; that is, $m|M$.

Note: for a single variable x_i, $\mathsf{ExistExtMon}(C, x_i)$ just tests if the partial derivative of p_C with respect to x_i is identically zero.

The following propositions list some of the known results used in the paper.

Proposition 1 ([7,8]). *Evaluating an arithmetic formula where the leaves are labelled* $\{-1, 0, 1\}$ *is in* DLOG *(even* GapNC^1*).*

Proposition 2 ([15]). *Given k ROFs in n variables, there is a deterministic (non black-box) algorithm that checks whether they sum to zero or not. The running time of the algorithm is* $n^{O(k)}$.

Proposition 3 (folklore). *The following problems are in* DLOG*:*

1) Given a formula F, a gate $g \in F$, and a variable x, check whether $x \in \mathsf{var}_g$.
2) Given a rooted tree T, and two nodes u, v, find lowest common ancestor (LCA) of u and v.

3 Read-Twice Formulas: Multilinearity and Identity Tests

In this section we consider the problem of testing multilinearity (MLIN) and testing identically zero (ACIT) on read-twice formulas. The individual degree of a variable in a polynomial p computed by read-twice formula F is bounded by two. Thus, multilinearity testing boils down to testing if the second order partial derivative of x_i is zero for every variable x_i. We use the inductive structure of a read-twice formula to test first order partial derivatives for zero, using the knowledge of MLIN and ACIT at gates at the lower levels, and to combine this information to compute MLIN and ACIT at the root.

Theorem 1. *For read-twice formulas, the problems* ACIT*,* MLIN*, and* $\mathsf{ExistExtMon}(\phi, x)$ *(where ϕ is the input formula and x is a single variable in it) are in P.*

Proof. Let ϕ be the given read-twice formula on variables $x_1, \ldots, x_n$, with S internal nodes. Without loss of generality, assume that ϕ is alternating; that is, inputs to a $+$ gate are either leaves or are $\times$ gates, and inputs to a $\times$ gate are either leaves or are $+$ gates.

We proceed by induction on the structure of the formula ϕ. For each gate g in ϕ and each variable $x \in X$, we iteratively compute the value of the constant term of p_g, denoted $\mathsf{const}(g)$, and the following set of 0-1 valued functions:

$$\mathsf{ACIT}(g) = 1 \Leftrightarrow p_g \equiv 0; \qquad \mathsf{MLIN}(g) = 1 \Leftrightarrow p_g \text{ is multilinear;}$$
$$\mathsf{ExistExtMon}(g, x) = 1 \Leftrightarrow p_g \text{ has a monomial } m \text{ that contains } x \text{ .}$$

Recall that $\mathsf{ExistExtMon}(g, x) = 1$ exactly when the partial derivative of p_g with respect to x is not identically zero; in this case we say that x survives in g. (Note: Since ϕ is a formula, the values $\mathsf{const}(g)$ can be represented with poly($|\phi|$) bits.)

The base case is when ϕ is a single variable or a constant. That is, ϕ consists of a single gate g, labelled $L \in X \cup \{0, +1, -1\}$. Then $\mathsf{ACIT}(g) = 1$ if and only if $L = 0$, $\mathsf{MLIN}(g) = 1$ always, and $\mathsf{ExistExtMon}(g, x) = 1$ if and only if $L = x$. Also, $\mathsf{const}(g)$ is L if $L \notin X$, 0 otherwise.

Now assume that for every gate u below the root gate of ϕ, the above functions have been computed and stored as bits. Let f be the root gate of ϕ. We show how to compute these functions at f. The order in which we compute them depends on whether f is $\times$ or a $+$ gate.

First, consider $f = g \times h$. We compute the functions in the order given below.

1. $\mathsf{const}(f) = \mathsf{const}(g) \times \mathsf{const}(h)$.
2. $\mathsf{ACIT}(f) = \mathsf{ACIT}(g) \vee \mathsf{ACIT}(h)$.
3. $\mathsf{MLIN}(f)$: If f is identically zero, then it is vacuously multilinear. Otherwise, for it to be multilinear, it must be the product of two (non-zero) multilinear polynomials in disjoint sets of variables. Thus

$$\mathsf{MLIN}(f) = \mathsf{ACIT}(f) \vee \Bigg[\mathsf{MLIN}(g) \wedge \mathsf{MLIN}(h) \wedge \left(\bigwedge_{x \in X} [\neg\mathsf{ExistExtMon}(g, x) \vee \neg\mathsf{ExistExtMon}(h, x)]\right)\Bigg]$$

   Note that the $\mathsf{ACIT}(f)$ term is necessary, since f can be multilinear even if, for instance, g is not, provided $h \equiv 0$.

4. $\mathsf{ExistExtMon}(f, x)$: x appears in p_f if and only if $p_f \not\equiv 0$ and x appears in at least one of p_g, p_h. Thus

$$\mathsf{ExistExtMon}(f, x) = \neg\mathsf{ACIT}(f) \wedge [\mathsf{ExistExtMon}(g, x) \vee \mathsf{ExistExtMon}(h, x)]$$

Next, consider $f = g + h$. We compute the functions in the order given below.

1. $\mathsf{const}(f) = \mathsf{const}(g) + \mathsf{const}(h)$.
2. $\mathsf{MLIN}(f)$: Since f is read-twice, a non-multilinear monomial in g cannot get cancelled by a non-multilinear monomial in h; that would require at least 4 occurrences of some variable. Thus, f is multilinear only if both g and h are. The converse is trivially true. Thus $\mathsf{MLIN}(f) = \mathsf{MLIN}(g) \wedge \mathsf{MLIN}(h)$.
3. $\mathsf{ExistExtMon}(f, x)$: This is the non-trivial part; we defer it to a bit later.
4. $\mathsf{ACIT}(f)$: Once we compute the functions above, this is straightforward:

$$\mathsf{ACIT}(f) = [\mathsf{const}(f) = 0] \wedge \bigwedge_{x \in X} \neg \mathsf{ExistExtMon}(f, x)$$

We now describe how to compute $\mathsf{ExistExtMon}(f, x)$ when $f = g+h$. If x survives in neither g nor h, then it does not survive in f. But if it survives in exactly one of g, h, it cannot get cancelled in the sum, so it survives in f. Thus

$$\mathsf{ExistExtMon}(g, x) \vee \mathsf{ExistExtMon}(h, x) = 0 \Longrightarrow \mathsf{ExistExtMon}(f, x) = 0$$
$$\mathsf{ExistExtMon}(g, x) \oplus \mathsf{ExistExtMon}(h, x) = 1 \Longrightarrow \mathsf{ExistExtMon}(f, x) = 1$$

So now assume that x survives in both g and h. We can write the polynomials computed at g, h as $p_g = \alpha x + \alpha'$ and $p_h = \beta x + \beta'$, where α', β' do not involve x; and we know that $\alpha \not\equiv 0$, $\beta \not\equiv 0$. We want to determine whether $\alpha + \beta \equiv 0$.

Since x appears in var_g and var_h, and since f is read-twice, we conclude that x is read exactly once each in g and in h. Hence α, β also do not involve x.

We construct a formula computing α as follows: In the sub-formula rooted at g, let ρ be the unique path from x to g. For each $+$ gate u on the path ρ, let u' be the child of u not on ρ; replace u' by the constant 0. Thus we retain only the parts that multiply x; that is, we compute αx. Setting $x = 1$ gives us a formula G computing α. A similar construction with the formula rooted at h gives a formula H computing β. Set $F = G + H$. Note that F is also a read-twice formula, and it computes $\alpha + \beta$. Thus in this case $\mathsf{ExistExtMon}(f, x) = 1 \Leftrightarrow \mathsf{ACIT}(F) = 0$, so we need to determine $\mathsf{ACIT}(F)$.

Let Y denote the set of variables appearing in F; $Y \subseteq X \setminus \{x\}$. Partition Y:
A: variables occurring only in G; B: variables occurring only in H;
C: variables occurring in G and H.

If $A \cup B = \emptyset$, then $Y = C$, and each variable in F appears once in G and once in H. That is, both G and H are read-once formulas. We can now determine $\mathsf{ACIT}(F)$ in time polynomial in the size of F using Proposition 2.

If $A \cup B \neq \emptyset$, then let $y \in A$. If y survives in G, it cannot get cancelled by anything in H, so it survives in F and $F \not\equiv 0$. Similarly, if any $y \in B$ survives in H, then $F \not\equiv 0$. We briefly defer how to determine this and complete the high-level argument. If no $y \in A$ survives in G, and no $y \in B$ survives in H, then let $F' = G' + H'$ be the formula obtained from F, G, H by setting variables in $A \cup B$ to 0. Clearly, the polynomial computed remains the same; thus $\alpha + \beta = p_F = p_F|_{A \cup B \leftarrow 0} = p_{F'}$. But F' satisfies the previous case (with respect to F', $A' \cup B' = \emptyset$), and so we can use Proposition 2 as before to determine $\mathsf{ACIT}(F') = \mathsf{ACIT}(F)$.

Now we describe how to determine whether a variable $y \in A$ survives in G. (The situation for $y \in B$ surviving in H is identical.) We exploit the special structure of G: there is a path ρ where all the $+$ gates have one argument 0 and the path ends in a leaf labeled 1. Let $\mathcal{T} = \{T_1, \ldots, T_\ell\}$ be the subtrees hanging off the $\times$ gates on ρ; let u_i be the root of T_i. Note that each $T_i \in \mathcal{T}$ is a sub-formula of our input formula ϕ, and hence by the iterative construction we know the values of the functions ACIT, MLIN, $\mathsf{ExistExtMon}$ at gates in these sub-trees. In fact, we already know that $\mathsf{ACIT}(u_i) = 0$ for all i, since we are in the situation where $\alpha \not\equiv 0$, and $\alpha = \prod_{i=1}^{\ell} p_{u_i}$. Hence, if y appears in just one sub-tree T_i, then $\mathsf{ExistExtMon}(G, y) = \mathsf{ExistExtMon}(u_i, y)$. If y appears in two sub-trees T_i, T_j, then $\mathsf{ExistExtMon}(G, y) = \mathsf{ExistExtMon}(u_i, y) \vee \mathsf{ExistExtMon}(u_j, y)$. □

A direct attempt to generalise this to read-k formulas would be to maintain $\mathsf{ExistExtMon}(f, x^i)$ for $1 \leq i \leq k$ at each gate. However, this does not work because in iteratively computing these values, we generate 2-sums of read-k formulas, not k-sums of ROFs, and cannot use Proposition 2.

4 Counting Monomials

We now consider the $\mathsf{MonCount}$ problem. In an ROF, a monomial, once generated in a sub-formula, can be cancelled only by multiplication with a zero polynomial. We exploit this fact to obtain an efficient algorithm for $\mathsf{MonCount}$ on ROFs. We then show that even for read-twice formulas, the problem becomes very hard. Since we cannot generalise Theorem 2 to read-twice formulas, we consider generalising it beyond ROFs to read-once BPs. For OBPs, similar properties as for ROFs hold, and again we obtain an efficient algorithm for $\mathsf{MonCount}$.

We start with ROFs:

Theorem 2. *Given a read-once formula F, $\mathsf{MonCount}(p_F)$ can be computed by an AC^0 circuit with oracle gates for GapNC^1 functions, and hence in DLOG.*

The following lemma is used in proving Theorem 2:

Lemma 1. *The language L defined below is in $\mathsf{C_{=}NC}^1$:*

$$L = \{\langle F, g\rangle \mid F \textit{ is an arithmetic formula, } g \textit{ is a gate in } F, \textit{ and } \mathsf{NZ}(g) = 0\}$$

For any polynomial p, $p \equiv 0$ if and only if the constant term of p is 0 and $\mathsf{MonCount}$(p) is 0. Hence, from Theorem 2 and Lemma 1 we have the following:

Corollary 1. *In the non-blackbox setting, ACIT on ROFs is in the GapNC hierarchy and hence in DLOG.*

Our next result shows that extending Theorem 2 to Read-k formulas for $k > 1$ is extremely unlikely. Even for formulas that are monotone (no negative constants) and read-twice, and furthermore, are decomposable as the sum of two read-once formulas, $\mathsf{MonCount}$ is at least as hard as $\#\mathsf{P}$.

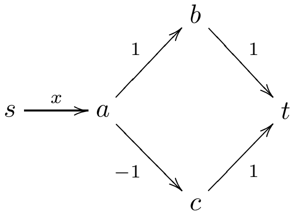
Theorem 3. $\mathsf{MonCount}$ *is* $\#\mathsf{P}$ *complete for the sum of two monotone read-once formulas.*

We now show how to count monomials in OBPs. The approach used in Theorem 2 does not directly generalize to OBPs, *i.e.,* knowing $\mathsf{MonCount}$ at immediately preceding nodes is not enough to compute $\mathsf{MonCount}$ at a given node in an OBP. However, since every variable occurs at most once in an OBP, every path generating a monomial should pass through one of these edges. This allows us to keep track of the monomials at any given node of the OBP, given the monomial count of all of its predecessors.

We begin with some notations. Let B be an occur-once BP on the set of variables X, and u, v be any nodes in B. Let $c(u, v)$ be the constant term in $p(u, v)$. We define the 0-1 valued indicator function that describes whether this term is non-zero:

$$\mathsf{NZ}(u, v) = \begin{cases} 1 & \text{if } c(u, v) \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

We cannot directly use the strategy we used for ROFs, since even in an OBP, there can be cancellations due to the constant terms. For instance, in the figure alongside, $\#p(s, b) = \#p(s, c) = 1$, but $\#p(s, t) = 0$. We therefore identify edges critical for a polynomial. We say that edge $e = (w, u)$ of B *is critical to* v if



1. $\mathsf{label}((w, u)) \in X$; and
2. B has a directed path ρ from u to v with all edges labeled by $\{-1, 1\}$.

We have the following structural property for the monomials in $p(s, v)$:

Lemma 2. *In an occur-once OBP B with start node s, for any node v in B,*

$$p(s, v) = c(s, v) + \sum_{(w,u) \text{ critical to } v} p(s, w) \cdot \mathsf{label}(w, u) \cdot c(u, v) \ .$$

Proof. Note that if edges $(w, u) \neq (w', u')$ are both critical to v, then the monomials in $p(s, w) \cdot \mathsf{label}(w, u)$ and $p(s, w') \cdot \mathsf{label}(w', u')$ will be disjoint, because P is occur-once. (The variables labeling (w, u) and (w', u') make the monomials distinct.) Moreover, for any monomial m in $p(s, v)$, there is exactly one critical edge (w, u) such that m has non-zero coefficient in the polynomial $p(s, w) \times \mathsf{label}(w, u)$. The critical edge corresponds to the last variable of the monomial to be "collected" en route to v from s. This completes the proof. □

For nodes w, u, v in B where (w, u) is an edge, define a 0-1 valued indicator function that specifies whether or not (w, u) is critical to v. That is,

$$\mathsf{critical}(\langle w, u \rangle, v) = \begin{cases} 1 & \text{if } (w, u) \text{ is critical for } v \\ 0 & \text{otherwise} \end{cases}$$

Using this and Lemma 2, we can show:

Lemma 3. *In an occur-once OBP B with start node s, for any node v in B,*

$$\#p(s,v) = \sum_{e=(w,u)} \mathsf{critical}(\langle w,u\rangle, v) \cdot \big(\#p(s,w) + \mathsf{NZ}(s,w)\big) \cdot \mathsf{NZ}(u,v).$$

If w is not in a layer to the left of v, then (w,u) cannot be critical to v, and so $\#p(s,w)$ is not required while computing $\#p(s,v)$. Hence we can sequentially evaluate $\#p(s,v)$ for all nodes v in layers going left to right, provided we have all the values $\mathsf{NZ}(s,w)$ and $\mathsf{critical}(\langle w,u\rangle, v)$.

Lemma 4. *Define languages L_1, L_2 as follows:*

$$L_1 = \{\langle B,u,v\rangle \mid B \textit{ is an OBP, } u,v \textit{ are nodes in } B, \textit{ and } \mathsf{NZ}(u,v) = 0.\}$$
$$L_2 = \left\{\langle B,u,v,w\rangle \mid \begin{array}{l} B \textit{ is an OBP, } u,v,w \textit{ are nodes in } B, \textit{ and} \\ \mathsf{critical}(\langle w,u\rangle, v) = 1. \end{array}\right\}$$

Then L_1 and L_2 are both in $\mathsf{C_{=}L}$.

From Lemma 3, the comment following it, and Lemma 4, we obtain a polynomial time algorithm to count the monomials in p_B. However, with a little bit of care, we can obtain the following stronger result:

Theorem 4. *Given an occur-once branching program B, the number of monomials in p_B can be computed in the GapL hierarchy and hence in NC^2.*

Proof. Starting from B, we construct another BP B' as follows: B' has a node v' for each node v of B. For each triple w,u,v where (w,u) is an edge in B, we check via oracles for L_1 and L_2 whether (w,u) is critical to v and whether $\mathsf{NZ}(u,v) = 1$. If both checks pass, we add an edge from w' to v'. We also check whether $\mathsf{NZ}(s,w) = 1$, and if so, we add an edge from s' to v'. (We do this for every w,u, so we may end up with multiple parallel edges from s' to v'. To avoid this, we can subdivide each such edge added.) B' thus implements the right-hand-side expression in Lemma 3. It follows that $p_{B'}(s',v')$ equals $\#p_B(s,v)$. Note that B' can be constructed in logspace with oracle access to $\mathsf{C_{=}L}$. Also, since B' is variable-free, it can be evaluated in GapL. Hence $\#p_B$ can be computed in the GapL hierarchy. □

As in Corollary 1, using Theorem 4 and Lemma 4, we have:

Corollary 2. *In the non-blackbox setting,* ACIT *on OBPs is in the* GapL *hierarchy and hence in NC^2.*

5 Zero-Test on a Monomial Coefficient (ZMC)

From [11], ZMC is known to be in the second level of CH and hard for the class $\mathsf{C_{=}P}$. For the case of multilinear BPs MBPs, we show that ZMC exactly characterizes the complexity class $\mathsf{C_{=}L}$.

Theorem 5. ZMC *for multilinear BPs is complete for* $\mathsf{C_=L}$*. More precisely,*

1. ZMC *for OBPs is hard for* $\mathsf{C_=L}$*.*
2. *Given a BP B computing a multilinear polynomial p_B, and given a multilinear monomial m, the coefficient of m in p_B can be computed in* GapL*.*

Proof. (Sketch) Hardness: A complete problem for $\mathsf{C_=L}$ is: does a BP B with labels from $\{-1,0,1\}$ evaluate to 0? Add a node t' as the new target node, and add edge $t \to t'$ labeled x to get B'. Then B' is an OBP, and $(B',x) \in \mathsf{ZMC}$ if and only if B evaluates to 0.

Upper bound: The idea is to construct (by relabelling the edges of B) a branching program B' computing a univariate polynomial, and a monomial m', such that the coefficients of m in p_B and of m' in $p_{B'}$ are the same. The coefficients of $p_{B'}$ can be computed in GapL, establishing the second statement. This will imply that the zero-test is in $\mathsf{C_=L}$. □

The upper bound above, for ZMC on MBPs, also applies to ROFs, since ROFs can be converted to OBPs by a standard construction. However, with a careful top-down algorithm, we can give a stronger upper bound of DLOG for ZMC on ROFs.

Theorem 6. *Given a read-once formula F computing a polynomial p_F, and given a multilinear monomial m, the coefficient of m in p_F can be computed in* DLOG*. Hence* ZMC *for ROFs is in* DLOG*.*

The lower bound proof in Theorem 5 can be modified to show that ZMC on ROFs is hard for $\mathsf{C_=NC^1}$. It is natural to ask whether there is a matching upper bound. In our construction above, we need to compute predicates of the form $[x \in \mathsf{var}_g]$. If these can be computed in $\mathsf{NC^1}$ for ROFs, then the monomial coefficients can be computed in $\mathsf{GapNC^1}$, improving the upper bound of ZMC to $\mathsf{C_=NC^1}$. However, this depends on the specific encoding in which the formula is presented. In the standard pointer representation, the problem models reachability in out-degree-1 directed acyclic graphs, and hence is as hard as DLOG.

6 Checking Existence of Monomial Extensions

We now address the problem $\mathsf{ExistExtMon}$. Given a monomial m, one wants to check if the polynomial computed by the input arithmetic circuit has a monomial M that extends m (that is, with $m|M$). This problem is seemingly harder than ZMC, and hence the bound of Theorem 5 does not directly apply to $\mathsf{ExistExtMon}$. We show that $\mathsf{ExistExtMon}$ for OBPs is in the GapL hierarchy.

Theorem 7. *The following problem lies in the* GapL *hierarchy: Given an occur-once branching program B and a multilinear monomial m, check whether p_B contains any monomial M such that $m|M$.*

The above bound can be brought down to DLOG for the case of ROFs.

Theorem 8. *The following problem is in* DLOG*: Given a read-once formula F computing a polynomial p_F, and given a multilinear monomial m, check whether p_F contains any monomial M such that $m|M$.*

7 Conclusion

In this paper, we studied the complexity of certain natural problems on severely restricted circuits.

We have shown that ACIT and MLIN are easy on read-twice formulas. In a recent extension, we have shown that using [5] instead of [15] yields a simpler algorithm that works even for read-3 formulas. Extending this to Read-k formulas for any constant $k > 3$ remains open.

We have shown that MonCount remains #P-hard for read-twice formulas.

We have shown that on read-once formulas and occur-once branching programs, the complexity of ZMC and ExistExtMon does reduce drastically. Ideally, we would like these problems to characterise complexity classes within P; we have partially succeeded in this.

We leave open the question of extending these bounds for formulas and branching programs that are constant-read.

References

1. Agrawal, M., Saha, C., Saptharishi, R., Saxena, N.: Jacobian hits circuits: Hitting-sets, lower bounds for depth-d occur-k formulas & depth-3 transcendence degree-k circuits. In: STOC (to Appear, 2012)
2. Agrawal, M., Vinay, V.: Arithmetic circuits: A chasm at depth four. In: FOCS, pp. 67–75 (2008)
3. Allender, E.: Arithmetic circuits and counting complexity classes. In: Krajicek, J. (ed.) Complexity of Computations and Proofs, Quaderni di Matematica, vol. 13, pp. 33–72. Seconda Universita di Napoli (2004)
4. Allender, E., Beals, R., Ogihara, M.: The complexity of matrix rank and feasible systems of linear equations. Computational Complexity 8(2), 99–126 (1999)
5. Anderson, M., van Melkebeek, D., Volkovich, I.: Derandomizing polynomial identity testing for multilinear constant-read formulae. In: CCC, pp. 273–282 (2011)
6. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press (2009)
7. Buss, S.: The Boolean formula value problem is in ALOGTIME. In: STOC, pp. 123–131 (1987)
8. Buss, S., Cook, S., Gupta, A., Ramachandran, V.: An optimal parallel algorithm for formula evaluation. SIAM Journal of Computation 21(4), 755–780 (1992)
9. Caussinus, H., McKenzie, P., Thérien, D., Vollmer, H.: Nondeterministic NC^1 computation. Journal of Computer and System Sciences 57, 200–212 (1998)
10. Datta, S., Mahajan, M., Rao, B.V.R., Thomas, M., Vollmer, H.: Counting classes and the fine structure between NC^1 and L. Theoretical Computer Science 417, 36–49 (2012)
11. Fournier, H., Malod, G., Mengel, S.: Monomials in arithmetic circuits: Complete problems in the counting hierarchy. In: STACS (to appear, 2012)
12. Jansen, M.J., Qiao, Y., Sarma, J.M.N.: Deterministic black-box identity testing π-ordered algebraic branching programs. In: FSTTCS, pp. 296–307 (2010)

13. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. Computational Complexity 13(1-2), 1–46 (2004)
14. Koiran, P., Perifel, S.: The complexity of two problems on arithmetic circuits. Theoretical Computer Science 389(1-2), 172–181 (2007)
15. Shpilka, A., Volkovich, I.: Read-once polynomial identity testing. In: STOC, pp. 507–516 (2008)

Fine and Wilf's Theorem and Pseudo-repetitions

Florin Manea[1,⋆], Robert Mercaş[2,⋆⋆], and Dirk Nowotka[1,⋆⋆⋆]

[1] Christian-Albrechts-Universität zu Kiel, Institut für Informatik,
D-24098 Kiel, Germany
{flm,dn}@informatik.uni-kiel.de

[2] Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik,
PSF 4120, D-39016 Magdeburg, Germany
robertmercas@gmail.com

Abstract. The notion of repetition of factors in words is central to considerations on sequences. One of the recent generalizations regarding this concept was introduced by Czeizler et al. (2010) and investigates a restricted version of that notion in the context of DNA computing and bioinformatics. It considers a word to be a pseudo-repetition if it is the iterated concatenation of one of its prefixes and the image of this prefix through an involution. We present here a series of results in the fashion of the Fine and Wilf Theorem in a more general setting where we consider the periods of some word given by a prefix of it and images of that prefix through some arbitrary morphism or antimorphism.

1 Introduction

The notions of repetition and primitivity are two fundamental concepts on words that play an important role in several research areas, e.g., stringology and algebraic coding theory. We call a word a repetition (or power) if there exists a decomposition of it in terms of one of its prefixes. This paper addresses combinatorial questions of a generalization of this concept, namely *pseudo-repetitions in words*. A word w is said to be a pseudo-repetition if it has a decomposition in terms of some prefix t and its image $f(t)$ under some morphism or antimorphism (for short "anti-/morphism") f, more precisely, $w \in t\{t, f(t)\}^+$.

A central combinatorial result regarding repetitions in words is the Fine and Wilf Theorem [1]. It states in a general context that if one can construct using two different words u and v two different sequences in such a way that one starts with u and the other with v, and they share a common prefix of at least the sum of the lengths of the two words minus their greatest common divisor, then the two sequences are equal. Moreover, u and v are both powers of a factor of length

[⋆] The work of Florin Manea is supported by the *DFG* grant 582014.

[⋆⋆] The work of Robert Mercaş is supported by the *Alexander von Humboldt Foundation.*

[⋆⋆⋆] The work of Dirk Nowotka is supported by the *DFG Heisenberg* grant 590179.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 668–680, 2012.

equal to the greatest common divisor of their lengths. This theorem addresses the probably most basic natural question one could ask about repetitions in sequences. Therefore, questions in the style of Fine and Wilf are considered whenever a new form of repetition in words is proposed. Up to now several generalizations of this theorem have been investigated [2–5]. We contribute to that line of research by stating Fine and Wilf style results in a more general setting where not only the words but also their images through some arbitrary functions are considered.

Having as a strong biological motivation the fact that Watson-Crick complementarity can be formalized as an antimorphic involution, and the fact that both a DNA-single stranded molecule and its complementary basically encode the same information, the authors of [5] introduce the notions of *pseudo-repetition* and *pseudo-primitivity*. In particular, a word is a *pseudo-repetition* if it can be expressed as the iterated concatenation between one of its prefixes and its image through a function f. A word is *pseudo-primitive* if it is not a pseudo-repetition. Until now, the considered functions were quite simple, being restricted to cases of anti-/morphic involutions, following the original motivation.

Considering that the notion of repetition is central in the study of combinatorics of words and the plethora of applications that this concept has in many parts of computer science, the study of pseudo-repetitions seems even more attractive, at least from a theoretical point of view. A natural extension of these results is to consider this concept for some more general classes of anti-/morphisms. Although the biological motivation seems appropriate only for the case of involutions, we are interested in identifying factors of words which can be written as the iterated concatenation of a word and its encoding through some arbitrary simple function f. In this setting, pseudo-repetitions can be seen as strings that have an intrinsic repetitive structure, hidden by rewriting some of the factors that define it through some anti-/morphism.

The next section presents some basis for the study of (pseudo-)repetitions and some basic observations that will help us throughout this work. In Section 3 we extend the pseudo-periodicity results obtained for involutory morphisms to arbitrary degree literal morphic functions. The section following it treats the antimorphic case. The results of both sections are shown to be optimal or, in one case, at least as good as the corresponding result for involutions. The paper is concluded with remarks stating the impossibility of deriving similar results for arbitrary anti-/morphisms.

We end this section with an overview of basic concepts used in this paper. For more detailed definitions we refer to [5, 6].

Let V be a finite alphabet. We denote by $\mathrm{alph}(w)$ the alphabet of letters that occur in a word $w \in V^*$ and by ε the *empty word*. The *length* of w is denoted by $|w|$. We say u is a *factor* of w, if $w = xuy$, for some words x, y. Moreover, u is a *prefix* of w, if $x = \varepsilon$ and a *suffix* of it if $y = \varepsilon$. Denote by $w[i]$ the symbol at position i in w, and by $w[i..j]$ the factor of w starting at position i and ending at position j, consisting of the concatenation of the symbols $w[i], \dots, w[j]$, where $1 \leq i \leq j \leq n$. Moreover, $w = u^{-1}v$, whenever $v = uw$. The powers of w are defined recursively by $w^0 = \varepsilon$, $w^n = ww^{n-1}$ for $n \geq 1$, and $w^\omega = ww\cdots$, an infinite concatenation of the word w. If w cannot be expressed as a power of another word, then w is said to be *primitive*.

We say that $f : V^* \to V^*$ is a morphism if $f(xy) = f(x)f(y)$ for any words $x, y \in V^*$. On the other hand, f is an antimorphism if $f(xy) = f(y)f(x)$. Note that, to define an anti-/morphism it is enough to give the definitions of $f(a)$ for all $a \in V$. For some anti-/morphism $f : V^* \to V^*$ we say that f is *uniform* if there exists a number k with $f(a) \in V^k$ for all $a \in V$. If $k = 1$, then f is called *literal*. If $f(a) = \varepsilon$ for some $a \in V$, then f is called *erasing*, otherwise *non-erasing*. Recall that an anti-/morphism $f : V^* \to V^*$ is an involution when $f^2(a) = a$ for all $a \in V$. Note that, all bijective anti-/morphisms are literal. Furthermore, a bijective morphism is also called *isomorphism*.

A word w is said to be an *f-repetition*, or, an f-power, if $w \in t\{t, f(t)\}^+$ for some prefix t of w. If w is not an f-power, then w is *f-primitive*.

The word *abcaab* is i-primitive, where i is the identical morphism, and f-primitive for some morphism or antimorphism f with $f(a) = b$, $f(b) = a$ and $f(c) = c$. However, for the morphism $f(a) = c$, $f(b) = a$ and $f(c) = b$ note that *abcaab* is the concatenation of *ab*, $f(ab) = ca$ and *ab*, thus, an f-power in this setting. In [5, 7] the authors investigate generalizations of the Fine and Wilf Theorem for f-repetitions, when f is a morphic or an antimorphic involution.

2 Fine and Wilf's Theorem and Pseudo-repetitions

The central concept of our investigation is periodicity with a main role played by the following result:

Theorem 1 (Fine and Wilf [1]). *Let u and v be in V^* and $d = \gcd(|u|, |v|)$. If two words $\alpha \in u\{u, v\}^*$ and $\beta \in v\{u, v\}^*$ have a common prefix of length greater than or equal to $|u| + |v| - d$, then u and v are powers of a common word of length d. Moreover, the bound $|u| + |v| - d$ is optimal.*

Other important parts in this work are played by functions, namely morphisms and antimorphisms of different types.

The study of generalizations of the Fine and Wilf theorem for the case of pseudo-repetitions, that is anti-/morphic involutions, started in [5] and was continued in [7]. The following summarize the existing results for pseudo-repetitions:

Theorem 2. *Let u and v be two words over an alphabet V and $f : V^* \to V^*$ a morphic involution. If $u\{u, f(u)\}^*$ and $v\{v, f(v)\}^*$ have a common prefix of length greater than or equal to $|u| + |v| - \gcd(|u|, |v|)$, then there exists $t \in V^*$ such that $u, v \in t\{t, f(t)\}^*$. Moreover, the bound $|u| + |v| - \gcd(|u|, |v|)$ is optimal.*

Theorem 3. *Let u and v be in V^* and $f : V^* \to V^*$ an antimorphic involution.*
1. If $|u| > |v| = 2\gcd(|u|, |v|)$ and $u\{u, f(u)\}^*$ and $v\{v, f(v)\}^*$ have a common prefix of length greater than or equal to $2|u| - \lfloor \gcd(|u|, |v|)/2 \rfloor$, then there exists $t \in V^*$ such that $u, v \in t\{t, f(t)\}^*$. The bound is optimal.

2. If $|u| > |v| > 2\gcd(|u|,|v|)$ *and* $u\{u, f(u)\}^*$ *and* $v\{v, f(v)\}^*$ *have a common prefix of length greater than or equal to* $2|u|+|v|-\gcd(|u|,|v|)-\lfloor \gcd(|u|,|v|)/2\rfloor$*, then there exists* $t \in V^*$ *such that* $u, v \in t\{t, f(t)\}^*$*.*

We start with the simple remark that for a two letter alphabet $\{a, b\}$, the case of bijective anti-/morphisms is quite trivial, since either f is the identity, or f is the involution given by $f(a) = b$ and $f(b) = a$. The results are given by Theorem 1 and its generalizations from [5, 7]. Thus, for the rest of this paper we consider alphabets of three or more letters.

Furthermore, since f is a bijective function from V to V, one can see f as a permutation of V. Thus, there exists a minimum $m > 0$ such that f^m is the identity of V. Generally, this value is denoted by $ord(f)$, called the order of f, and is less than $g(|V|)$, where g is the Landau function[1].

We end this section with a well known lemma and an immediate observation that will help us with the proofs throughout the paper:

Lemma 1. *For a word* w*, if* $ww = xwy$ *with* $x \neq \varepsilon$ *and* $y \neq \varepsilon$*, then* x*,* y *and* w *are powers of the same word* t*.*

Lemma 2. *Let* $w \in V^*$ *be a word and* $f : V^* \to V^*$ *a bijective anti-/morphism. If* $w = f(w)$*, then, for any letter* $a \in alph(w)$*, we have* $f^2(a) = a$*.*

Proof. Let us denote $w = a_1 \cdots a_n$ with $a_i \in V$, where $1 \leq i \leq n$. Since $f(w) = w$, then $f^2(w) = f(f(w)) = f(w)$, and it follows that $w = a_1 \cdots a_n = f^2(a_1) \cdots f^2(a_n)$. Thus, $a_i = f^2(a_i)$ for all i with $1 \leq i \leq n$. □

3 Morphisms and Pseudo-repetitions

Using standard techniques similar to the one in [8] one can prove the following first important result:

Theorem 4. *Let* $u, v \in V^*$ *and* $f : V^* \to V^*$ *be an isomorphism with* $ord(f) = k + 1$*. If a word* $\alpha \in u\{u, f(u), \ldots, f^k(u), v, f(v), \ldots, f^k(v)\}^*$ *has a common prefix of length greater than or equal to* $|u| + |v| - \gcd(|u|,|v|)$ *with a word* $\beta \in v\{u, f(u), \ldots, f^k(u), v, f(v), \ldots, f^k(v)\}^*$*, then there exists a* $t \in V^*$*, such that* $u, v \in t\{t, f(t), \ldots, f^k(t)\}^*$*.*

Proof. Note that, if $|u| = |v|$ then $u = v$ and the claim follows trivially. Assume without loss of generality that $|u| > |v|$. Then for some word w we have $u = vw$. Observe that the prefix of length $|v|$ of $v^{-1}\beta$ is an iteration of $f(v)$. Denoting

[1] The Landau function is defined for every natural number n as the largest order of an element in the symmetric group S_n. Equivalently, $g(n)$ is the largest least common multiple of any partition of n, or the maximum number of times a permutation of n elements can be recursively applied to itself before it returns to its starting sequence. It is known that $\lim_{n\to\infty} \frac{\ln(g(n))}{\sqrt{n \ln(n)}} = 1$.

this prefix by z and changing appropriately all occurrences from α and β of iterations of f over v with iterations over z, we get

$$v^{-1}\alpha \in w\{w, f(w), \ldots, f^k(w), z, f(z), \ldots, f^k(z)\}^*$$

and

$$v^{-1}\beta \in z\{w, f(w), \ldots, f^k(w), z, f(z), \ldots, f^k(z)\}^*$$

and the claim follows by induction. □

This generalizes both Fine and Wilf and Kari et al. periodicity results.

Corollary 1. *Let $u, v \in V^*$ and $f : V^* \to V^*$ be an isomorphism with $ord(f) = k+1$. If $u\{u, f(u), \ldots, f^k(u)\}^*$ and $v\{v, f(v), \ldots, f^k(v)\}^*$ have a common prefix of length greater than or equal to $|u|+|v|-\gcd(|u|, |v|)$, then there exists $t \in V^*$, such that $u, v \in t\{t, f(t), \ldots, f^k(t)\}^*$. The bound is optimal.*

Next we show that in the case of arbitrary bijective literal morphisms the result of Theorem 4 is optimal also regarding the number of different iterations of the function f that are used in expressing both u and v. The counterexample obtained in this result exploits the algebraic properties of f, as permutation.

Proposition 1. *Let $f : V^* \to V^*$ be an isomorphism with $ord(f) = k+1$. There exist $u, v \in V^*$ with $|u| = |v| + \gcd(|u|, |v|)$ and $vf(v)$ a prefix of u^2, such that u is not part of $t\{f^{i_1}(t), \ldots, f^{i_\ell}(t)\}^*$ for any common prefix t of u and v with $v \in t\{t, f(t), \ldots, f^k(t)\}^*$, and $\{i_1, \ldots, i_\ell\}$ a set strictly included in $\{1, \ldots, k\}$.*

Proof. Let us assume that $V = \{a_1, \ldots, a_n\}$. As we explained, f is seen as a permutation of V. Assume that f has m disjoint cycles and let $c_i = (a_{i,1}, \ldots, a_{i,p_i})$ for $1 \leq i \leq m$ denote these cycles (we assume that the numbers in a cycle are ordered increasingly). Also let x_i be the word obtained by concatenating the letters $a_{i,j}$ of a cycle for $1 \leq j \leq p_i$ and denote $x = x_1 \ldots x_m$. Now take

$$u = xf^k(x)f^{k-1}(x)\cdots f(x) \text{ and } v = xf^k(x)f^{k-1}(x)\cdots f^2(x),$$

where u basically contains all possible iterations of f, while v contains only k factors. Note that $\gcd(|u|, |v|) = |x|$ and that $|u| = |v| + |x|$. It is straightforward to check that $vf(v)$ is a prefix of length $|u| + |v| - |x|$ of u^2.

Now we show that there does not exist a word t, such that

$$u \in t\{f^{i_1}(t), \ldots, f^{i_\ell}(t)\}^* \text{ and } v \in t\{t, f(t), f^2(t), \ldots, f^k(t)\}^*$$

for any set $\{i_1, \ldots, i_\ell\}$ strictly included in $\{1, \ldots, k\}$.

If such a word t exists, then its length is a divisor of n (as it divides both $|u| = (k+1)n$ and $|v| = kn$). If $|t| = n$ one would not be able to generate all the factors of length n of u using only the factors $f^{i_1}(t), \ldots, f^{i_\ell}(t)$, as the order of f is $k+1 > \ell$. If $|t| < n$, then $x = tf^{j_1}(t) \ldots f^{j_p}(t)$ for a set of numbers $\{j_1, \ldots, j_p\}$ included in $\{i_1, \ldots, i_\ell\}$. Let us assume that f is not a cyclic permutation. If t does not contain any symbol of x_m, then these symbols do not appear in $f^\ell(t)$

for any ℓ, thus a contradiction with the fact that $x = tf^{j_1}(t) \dots f^{j_p}(t)$. Hence, t has as suffix a part of x_m and $f^{j_p}(t)$ is included in x_m; from this we get that t contains only symbols from x_m, another contradiction. It follows that f is a cyclic permutation (thus, of order n) and that all the factors of length n of u begin with a different letter. Therefore, all iterations of f must be used in writing u as the catenation of factors of the form $f^i(t)$. □

Following the results of Kari et al. a natural question that comes up is what are good bounds for the case when we consider descriptions given by some prefix of the words and applications of a morphism to that prefix. The rest of this section is dedicated to finding such optimal bounds.

Example 1. Let i be a natural number. Consider the words

$$u = b^i da^i ca^i e \text{ and } v = b^i da^i c,$$

and an isomorphism f with $f(a) = b$, $f(b) = a$, $f(c) = d$, $f(d) = e$ and $f(e) = c$. The words u^2 and $vf(v)^2$ share a prefix of length $|u| + |v| - 1$ and no word t exists, such that $u, v \in t\{t, f(t)\}^*$. □

Proposition 2. *Let $u, v \in V^*$ such that $|u| > |v| = 2\gcd(|u|, |v|)$ and $f : V^* \to V^*$ be an isomorphism. If $\alpha \in u\{u, f(u)\}^*$ and $\beta \in v\{v, f(v)\}^*$ have a common prefix of length greater than or equal to $|u| + |v|$, then there exists $t \in V^*$, such that $u, v \in t\{t, f(t)\}^*$. The bound is optimal.*

Proof. Let v_1 be the prefix of length $\gcd(|u|, |v|)$ of v, where $v = v_1 v_2$. It is rather easy to see that $u \in v\{v, f(v)\}^* v_1$ or $u \in v\{v, f(v)\}^* f(v_1)$.

When u ends with v_1, it follows that v_2 is a prefix of u or $f(u)$, since the first u of α is followed by either u or $f(u)$. In the first case, v_2 is a prefix of v and, thus $v_1 = v_2$. In the second case, we have $v_2 = f(v_1)$. Moreover, looking at what follows v_2 in β, either $f(v_2) = v_1$ or $f(v_2) = f(v_1)$. In both cases, one may take $t = v_1$ and obtain that $u, v \in t\{t, f(t)\}^*$.

Let us now analyse the case when u ends with $f(v_1)$. Here, we obtain as above, that $f(v_2)$ is either a prefix of u or of $f(u)$. First, we obtain that $f(v_2) = v_1$, and, looking at what follows the prefix $uf(v_2)$ of β we once more get that $v_2 \in \{v_1, f(v_1)\}$. Similarly, in the second case, $f(v_2) = f(v_1)$, thus, $v_2 = v_1$. In both cases, one may take $t = v_1$ and obtain that $u, v \in t\{t, f(t)\}^*$. The conclusion follows with the optimality derived from Example 1. □

However, when the length of the shortest word is strictly greater than two times the greatest common divisor of the two words, the result is a bit more complicated. Considering that f is a permutation, and taking into account again the algebraic properties that follow from this, we get the following results.

Proposition 3. *Let $u, v \in V^*$ such that $|u| > |v| > 2\gcd(|u|, |v|)$, and $f : V^* \to V^*$ be an isomorphism. If $\alpha \in uu\{u, f(u)\}^*$ and $\beta \in v\{v, f(v)\}^*$ have a common prefix of length greater than or equal to $2|u|$, then there exists $t \in V^*$, such that $u, v \in t\{t, f(t)\}^*$. The bound is optimal.*

Proof. Denote by u' the longest prefix of u with $u' \in v\{v, f(v)\}^*$ and by v_1 the prefix of v with $|v_1| = |u| - |u'|$. Obviously, $\gcd(|v_1|, |v|) = d \neq |v|/2$.

Let us assume first that $|v_1| < |v|/2$ and denote $v = v_1v_2v_3$, where $|v_2| = |v_1|$.

Consider the case when $\alpha = u'v_1u\alpha' = u'v_1v_1v_2v_3u''\alpha'$, where $\alpha' \in \{u, f(u)\}^*$ and $u = vu''$. Note that u' is a prefix of β, such that $\beta = u'v\beta'$, with $\beta' \in \{v, f(v)\}^*$. The discussion follows now several cases.

If $\beta = u'vv\beta''$, then since the factors v_1v following u' in α and vv_1 following u' in β match, by Lemma 1 we obtain that v_1 and v are both powers of the same word t. Thus, we easily get that $u, v \in t\{t, f(t)\}^*$.

Now take $\beta = u'vf(v)\beta''$. We have $v_2 = v_1$ and $v_3 = v_1^\ell x$ for some number $\ell \geq 0$ and $x \in V^*$ a possibly empty prefix of v_1 with $|x| < |v_1|$. Denoting $v_1 = xy$ we obtain that $yx = f(v_1)$. If u'' starts with v we have the prefix yxv_1 of yxu'' equal to the prefix $f(v_1)f(v_1)$ of β'. Therefore, $f(v_1) = v_1$. It follows that f is the identity on the alphabet of the words u and v, and the conclusion follows from Theorem 1. If u'' starts with $f(v)$, this $f(v)$ ends with the suffix $f(yx)$. But this suffix matches either a prefix $f(v_1)$ of β'' or a prefix v_1 of β''. In the first case we get that f is the identity on the alphabet of u and v, and we conclude by Theorem 1, while in the second case we get that $f^2(v_1) = v_1$, and, thus, f is an involution on the alphabet of u and v, and conclude by Theorem 2.

Next, we analyse the case when $\alpha = u'f(v_1)u\alpha' = u'f(v_1)v_1v_2v_3u''\alpha'$, where $\alpha' \in \{u, f(u)\}^*$ and $u = vu''$. Note that u' is a prefix of β such that $\beta = u'f(v)\beta'$ with $\beta' \in \{v, f(v)\}^*$. Here $f(v_2) = v_1$ and the suffix $f(v_1)$ of the u factor occurring before α' in α matches an $f(v_2)$ or a v_2 factor from β. In the first case we obtain that f is the identity on all letters of u and v, and and we conclude by Theorem 1, while in the second case we get that $f^2(v_1) = v_1$ and, thus, f is an involution on the alphabet of u and v, and we conclude by Theorem 2.

We move now to the case when $|v_1| > |v|/2$ and set $v = v_1v_2$ with $|v_2| < |v_1|$.

Assume first that $\alpha = u'v_1u\alpha' = u'v_1v_1v_2u''\alpha'$, where $\alpha' \in \{u, f(u)\}^*$ and $u = vu''$. Note that u' is a prefix of β such that $\beta = u'v\beta'$ with $\beta' \in \{v, f(v)\}^*$. Clearly, v_2 is a prefix of v_1. If β' starts with v, then by Lemma 1 both v_1 and v are powers of some t, and, therefore, u and v are in $t\{t, f(t)\}^*$. If β' starts with $f(v)$, then $f(v_1)$ has v_2 as a suffix.

If u'' starts with v we obtain that the suffix $f(v_2)$ of the prefix $f(v)$ of β' matches the prefix v_2 of the prefix v of u''. Thus, f is the identity on the symbols of v_2. It is easy to see that the symbols of v_1 are those of v_2 and $f(v_2)$, and so, f is the identity also for the symbols of v_1 and, consequently, for the symbols of u and v. The conclusion follows from Theorem 1.

Now, consider the case when u'' starts with $f(v)$. If β' starts with $f(v)f(v)$ we obtain that $f(v_2)$ is a suffix of $f(v_1)$ and, thus, it is equal to v_2. As in the previous case, this leads to the conclusion that f is the identity on the alphabet of u and v, and the conclusion follows from Theorem 1. If β' starts with $f(v)v$ we obtain that $f(v_2)$ is a suffix of v_1 and, thus, $f^2(v_2)$ is a suffix of $f(v_1)$. Therefore, f is an involution on the alphabet of v_2 and an involution on the alphabet of u and v. The conclusion follows from Theorem 2.

Assume now that $\alpha = u'f(v_1)u\alpha' = u'f(v_1)v_1v_2u''\alpha'$, where $\alpha' \in \{u, f(u)\}^*$ and $u = vu''$. Note that u' is a prefix of β such that $\beta = u'f(v)\beta'$ with $\beta' \in \{v, f(v)\}^*$ and $f(v_2)$ is a prefix of v_1.

Assume first that β' starts with $f(v)$. If u'' starts with $f(v_1)$, then $f(v_2)$ is a prefix of $f(v_1)$. But $f^2(v_2)$ is a prefix of $f(v_1)$ as well, so f is the identity on v_2. As in the previous cases, we obtain that f is the identity on all letters of u and v, and with the help of Theorem 1 reach the conclusion.

When u'' starts with v, if β' starts with $f(v)v$ we have that $v_1v_2 = f(v_2)f(v_1)$ and $v_1v_2 = f(v_2)v_1$. Thus, $v_1 = f(v_1)$ and f is the identity for the alphabet of u and v. The conclusion follows again from Theorem 1. If β' starts with $f(v)f(v)$ we get that u'' starts with either vv or with $vf(v_1)$. In the latter case the conclusion follows as in the case when u'' starts with $f(v_1)$. In the first case, the analysis is restarted, ending up with either a solution as in the case when β' starts with $f(v)v$, or the case when u'' starts with $f(v_1)$, as u ends with $f(v_1)$. Hence, we conclude that this case leads also to what we wanted to prove.

Finally, assume that β' starts with v. If u'' starts with v_1 we obtain that both $f(v_2)$ and v_2 are prefixes of v_1, so f is the identity on the alphabet of u and v. If u'' starts with $f(v_1)$, then $f(v_1)$ starts with v_2, so $f^2(v_2) = v_2$. Thus, f is an involution on the alphabet of u and v, and we conclude by Theorem 2.

The optimality of the result is obtained from Example 2. □

Example 2. Let i be a natural number. Consider the words

$$u = (deadebdec)^i dec \text{ and } v = (deadebdec)^i,$$

and an isomorphism f with $f(a) = c$, $f(b) = a$, $f(c) = b$, $f(d) = d$ and $f(e) = e$. The words u^2 and $vf(v)^2$ share a common prefix of length $2|u| - 1$ and no word t exists, such that $u, v \in t\{t, f(t)\}^*$. □

Using the strategy of the proof of Proposition 3, one gets the following result:

Proposition 4. *Let $u, v \in V^*$ such that $|u| > |v| > 2\gcd(|u|, |v|)$ and $f : V^* \to V^*$ be an isomorphism. If $\alpha \in uf(u)\{u, f(u)\}^*$ and $\beta \in v\{v, f(v)\}^*$ have a common prefix of length greater than or equal to $2|u| + \gcd(|u|, |v|)$, then there exists $t \in V^*$ such that $u, v \in t\{t, f(t)\}^*$. The bound is optimal.*

Proof. Denoting again by u' the longest prefix of u with $u' \in v\{v, f(v)\}^*$, for a factorization $v = v_1 \cdots v_m$ with $|v_i| = \gcd(|u|, |v|) = d$ for all $1 \leq i \leq m$ we let $v' = v_1 \cdots v_i$ be the prefix of v for which $|v'| = |u| - |u'|$. It is straightforward that $\gcd(|v'|, |v|) = \gcd(|u|, |v|) = d \neq |v|/2$, so $\gcd(i, m) = 1$.

The proof consists of several case analysis just as that of Proposition 3.

First we consider $\alpha = u'v_1 \dots v_i f(u)\alpha'$, where $\alpha' \in \{u, f(u)\}^*$, and thus $\beta = u'v\beta'$ with $\beta' \in \{v, f(v)\}^*$, and analyze what happens when $i < m/2$ (we have in this case $f(v_1) = v_{i+1}$ and $f^2(v_1) = f(v_{i+1}) = v_{2i+1}$), and then what happens when $i > m/2$ (now we have $2i + 1 > m$ and $f^2(v_1) = f(v_{i+1}) \in \{v_{(2i+1) \bmod m}, f(v_{(2i+1) \bmod m})\}$).

Finally, we consider the case when $\alpha = u'f(v_1 \dots v_i)f(u)\alpha'$, where $\alpha' \in \{u, f(u)\}^*$ and $u = vu''$.

The optimality of the result is obtained from Example 3. □

Example 3. Let i be a natural number. Consider the words

$$u = (abcabdabe)^i abc \text{ and } v = (abcabdabe)^i$$

and an isomorphism f with $f(a) = a$, $f(b) = b$, $f(c) = d$, $f(d) = e$ and $f(e) = c$. The words $uf(u)ab$ and v^3 share a common prefix of length $2|u| + \gcd(|u|, |v|) - 1$ and no word t exists such that $u, v \in t\{t, f(t)\}^*$. □

4 Pseudo-repetitions for Antimorphisms

For a bijective antimorphism f and a word t, denote by $f^{-1}(t)$ the unique word x with $f(x) = t$. Clearly, $f^{2ord(f)-1}(t) = f^{-1}(t)$, as $f^{2ord(f)}(x) = x$, but not necessarily $f^{ord(f)}(x) = x$, as for some even integer k, $f^{k+1}(x)$ is x mirrored.

First, we note that a result similar to that of Theorem 4 does not hold in this case, even when we allow common prefixes of arbitrarily large length.

Example 4. Let i be a natural number. Consider the words

$$u = a^i b^i c \text{ and } v = a^i b^i,$$

and a bijective antimorphism f with $f(a) = e$, $f(b) = d$, $f(c) = c$. Moreover, f can be chosen as involution. The infinite word $w = a^i b^i c (d^i e^i)^\omega$ can be written as $w = uf(v)^\omega = vf(u)f(v)^\omega$ and all three words u, v and w are f-primitive. □

So which are the bounds in the antimorphism case? When $|v| = 2\gcd(|u|, |v|)$ the following result is not difficult to prove:

Proposition 5. *Let $u, v \in V^*$ with $|u| > |v| = 2\gcd(|u|, |v|)$ and $f : V^* \to V^*$ be a bijective antimorphism. If $\alpha \in u\{u, f(u)\}^*$ and $\beta \in v\{v, f(v)\}^*$ have a common prefix of length greater than or equal to $2|u| - \lfloor \gcd(|u|, |v|)/2 \rfloor$, then there exists $t \in V^*$ such that $u, v \in t\{t, f(t)\}^*$ or $u, v \in t\{t, f^{-1}(t)\}^*$. The bound is optimal.*

Proof. Since $|v| = 2\gcd(|u|, |v|)$, there exist factorizations $v = v_1 v_2$ and $u = v_1 v_2 \dots v_{2k+1}$, where $k \geq 1$ and $|v_i| = \gcd(|u|, |v|)$ for $1 \leq i \leq 2k + 1$.

Assume first that $u \in v\{v, f(v)\}^* v_1$, thus the prefix of length $|u|$ of β is followed by v_2. If uu is a prefix of α, then $v_1 = v_{2k+1}$, $v_2 = v_1$ and $u \in v_1\{v_1, f(v_1)\}^+$. If $uf(u)$ is a prefix of α, then $v_1 = v_{2k+1}$, $v_2 = f(v_{2k+1})$ and, thus, $v_2 = f(v_1)$. When $u \in \{v\}^+ v_1$ we have $u \in v_1\{v_1, f(v_1)\}^+$. If exists i such that $1 < i \leq k$ and $v_{2i-1} v_{2i} = f(v_2) f(v_1)$, we look at the factor that corresponds to $f(v_{2i}) f(v_{2i-1})$ in the occurrence of $f(u)$ of the prefix $uf(u)$ of α that we analyse; note that $2i \leq 2|u| - \lfloor \gcd(|u|, |v|)/2 \rfloor$. We have $f(v_{2i}) f(v_{2i-1}) \in \{v_1 v_2, f(v_2) f(v_1)\}$. In both cases we have that f is an involution and the conclusion follows by Theorem 3.

Now assume that $u \in v\{v, f(v)\}^* f(v_2)$, that is the prefix of length $|u|$ of β is followed by $f(v_1)$. If uu is a prefix of α, then $f(v_2) = v_{2k+1}$ and $f(v_1) = v_1$. Looking at the prefix of length $|v|$ of the second occurrence of u in α we obtain that $v_2 = f(v_2)$ or $v_2 = v_1$. In the first case, f is an involution and we conclude by Theorem 3, while in the second case we have $u \in v_1\{v_1, f(v_1)\}^+$. If $uf(u)$ is a prefix of α, then $f(v_2) = v_{2k+1}$ and $f(v_1) = f(v_{2k+1})$, and, thus, $f(v_2) = v_1$. As above, if $u \in \{v\}^+ f(v_2)$, then $u \in v_1\{v_1, f^{-1}(v_1)\}^+$. If there exists i such that $1 < i \leq k$ and $v_{2i-1}v_{2i} = f(v_2)f(v_1)$ we look at the factor that corresponds to $f(v_{2i})f(v_{2i-1})$ in the occurrence of $f(u)$ of the prefix $uf(u)$ of α that we analyse; note that $2i \leq 2|u| - \lfloor \gcd(|u|, |v|)/2 \lfloor$. The conclusion follows as above.

In conclusion, there always exists a prefix t of u such that $u, v \in t\{t, f(t)\}^*$ or $u, v \in t\{t, f^{-1}(t)\}^*$. The optimality of the bound $2|u| - \lfloor \gcd(|u|, |v|)/2 \lfloor$ follows from the optimality result in Theorem 3. □

In fact, the following example shows that there are words u and v as in the statement of the previous proposition for which there exists a unique t such that $u, v \in t\{t, f(t)\}^*$ (or, alternatively, $u, v \in t\{t, f^{-1}(t)\}^*$).

Example 5. This example shows that for any bijective antimorphism $f : V^* \to V^*$ which is not an involution there exist two words $u, v \in V^*$ such that $|u| > |v| = 2\gcd(|u|, |v|)$ and the words $\alpha \in u\{u, f(u)\}^*$ and $\beta \in v\{v, f(v)\}^*$ having a common prefix of length greater than or equal to $2|u| + \gcd(|u|, |v|)$ such that there exists a unique prefix x of v such that $u, v \in x\{x, f^{-1}(x)\}^*$ and there exists no prefix t of v such that $u, v \in t\{t, f(t)\}^*$.

Since f is not an involution, f has at least one cycle of length greater than or equal to 3; denote the elements of this cycle with $a_1, a_2, \ldots, a_k$ with $k \geq 3$, $f(a_i) = a_{i+1}$ for $1 \leq i \leq k-1$ and $f(a_k) = a_1$. Consider the words

$$u = a_1 a_k a_{k-1} \ldots a_3 a_2 a_1 a_2 \ldots a_{k-1} a_k a_1 a_k a_{k-1} \ldots a_3 a_2$$

and

$$v = a_1 a_k a_{k-1} \ldots a_3 a_2 a_1 a_2 \ldots a_{k-1} a_k.$$

The words $uf(u)$ and $vf(v)^2$ are equal, but no word t exists such that u and v are both in $t\{t, f(t)\}^*$. Clearly, an infinite iteration of $uf(u) = vf(v)^2$ still has two different factorizations: one as a word from $u\{u, f(u)\}^*$ and one from $v\{v, f(v)\}^*$, respectively. Also, $u = xf^{-1}(x)x$ and $v = xf^{-1}(x)$, for $x = a_1 a_k a_{k-1} \ldots a_3 a_2$, and there is no other prefix t of u and v such that $u, v \in t\{t, f^{-1}(t)\}^*$.

Similar examples can be devised to show that, for any bijective antimorphism $f : V^* \to V^*$, there exist two words $u, v \in V^*$ with $|u| > |v| = 2\gcd(|u|, |v|)$ and the words $\alpha \in u\{u, f(u)\}^*$ and $\beta \in v\{v, f(v)\}^*$ having a common prefix of length greater than or equal to $2|u| + \gcd(|u|, |v|)$ such that there exists a unique prefix x of v such that $u, v \in x\{x, f(x)\}^*$ and there exists no prefix t of v such that $u, v \in t\{t, f^{-1}(t)\}^*$. Just take, in the above setting,

$$u = a_1 a_k a_{k-1} \ldots a_3 a_2 a_3 a_4 \ldots a_k a_1 a_2 a_k a_{k-1} \ldots a_3 a_2$$

and

$$v = a_1 a_k a_{k-1} \ldots a_3 a_2 a_3 a_4 \ldots a_k a_1 a_2.$$

If f is an involution, then we have $f^{-1}(x) = f(x)$ for any word x. Assume that f is over an alphabet including $\{a, b\}$, with $f(a) \notin \{a, b\}$. Let i be a prime number, and consider the words $u = (ab)^i f((ab)^i)(ab)^i$ and $v = (ab)^i f((ab)^i)$. As in the previous cases, $uf(u) = v(f(v))^2$ and $u, v \in x\{x, f(x)\}^*$ for $x = (ab)^i$, but there is no other prefix t of u and v such that $u, v \in t\{t, f(t)\}^*$. □

The following result represents a variation of Lemma 1. The proof is done identifying factors that give equalities as in Lemma 2 and conclude that the antimorphism is an involution.

Lemma 3. *For a word w and a bijective antimorphism f defined on the alphabet of w, if w or $f(w)$ are proper factors of $\{w, f(w)\}^2$, such that not all three factors are equal, it is the case that f is an involution.*

Proof. Assume first that $w = a_1 \cdots a_n$ is a proper factor of $wf(w)$, where n is the length of w. It follows that for some j with $1 < j \leq n$ we have that $a_j \cdots a_n = f(a_n) \cdots f(a_j)$ and by Lemma 2 we get that for the alphabet of this factor, f is an involution. Looking now at the equality $a_1 \cdots a_{j-1} = a_{n-j+1} \cdots a_n$, one can easily prove that the alphabet of this factor is the same as the one of $a_j \cdots a_n$, and, therefore, f is an involution for all letters in w.

If w is a proper factor of $f(w)w$, then $a_1 \cdots a_j = f(a_j) \cdots f(a_1)$ and, again by Lemma 2, for the alphabet of this factor f is an involution. The equality $a_{j+1} \cdots a_n = a_1 \cdots a_{n-j}$ shows that f is an involution for w.

If w is a proper factor of $f(w)f(w)$, then $a_1 \cdots a_j = f(a_j) \cdots f(a_1)$ and $a_{j+1} \cdots a_n = f(a_n) \cdots f(a_{j+1})$. Again by Lemma 2, we conclude that f is an involution for w.

Assume that $f(w)$ is a proper factor of $wf(w)$. It follows that for some j with $1 < j \leq n$ we have that $f(a_n) \cdots f(a_j) = a_j \cdots a_n$, and by Lemma 2 for the alphabet of this factor f is an involution. From the equality $f(a_{j-1}) \cdots f(a_1) = f(a_n) \cdots f(a_{n-j+1})$, one can prove that the alphabet of this factor is the same as that of $a_j \cdots a_n$, and so f is an involution for all letters in w.

If $f(w)$ is a proper factor of $f(w)w$, then $f(a_j) \cdots f(a_1) = a_1 \cdots a_j$ and by Lemma 2 for the alphabet of this factor f is an involution. From the equality $f(a_n) \cdots f(a_{j+1}) = f(a_{n-j}) \cdots f(a_1)$, one concludes again that f is an involution for the entire alphabet of w.

Finally, take $f(w)$ a proper factor of ww. Since $f(a_n) \cdots f(a_j) = a_j \cdots a_n$ and $f(a_{j-1}) \cdots f(a_1) = a_1 \cdots a_{j-1}$, by Lemma 2 we conclude that f is an involution for $\text{alph}(w)$. □

The case of $|v| \geq 3 \gcd(|u|, |v|)$ is proved by looking at the alignment of the prefix v, or, respectively, suffix $f(v)$, of the second factor of length $|u|$ of α with the corresponding factors from β.

Proposition 6. *Let $u, v \in V^*$ be such that $|u| > |v| > 2 \gcd(|u|, |v|)$ and let $f : V^* \to V^*$ be a bijective antimorphism. If $\alpha \in u\{u, f(u)\}^*$ and $\beta \in v\{v, f(v)\}^*$ have a common prefix of length greater than or equal to $2|u| + |v| - \gcd(|u|, |v|) - \lfloor \gcd(|u|, |v|)/2 \rfloor$, then there exists $t \in V^*$, such that $u, v \in t\{t, f(t)\}^*$.*

Proof. Let $d = \gcd(|u|, |v|)$. The proof of this is based on the key remark that the prefix u in α is followed by either v, the prefix of u, or $f(u)$, which has $f(v)$

as a suffix. Further, it is worth noting that, when α has $uf(u)$ as a prefix, the suffix $f(v)$ of $f(u)$ is a proper factor of a word from $\{v, f(v)\}^2$. This is true since, otherwise, we have that for some coprime integers k, k' with $|u| = kd$ and $|v| = k'd \geq 3d$ there exists an integer h such that $2kd = hk'd$. Thus, from $2k = hk'$ and the fact that k and k' are coprime, we get that $k = h$ and $k' = 2$, a contradiction.

Now, in both cases above, since $|v| \geq 3d$, we get that there exist the words $x, y, z \in \{v, f(v)\}$ such that x is a proper factor of yz. If not all three factors x, y, and z are equal, we conclude by Lemma 3. Otherwise, we have $x = y = z = v$, and it follows by Lemma 1 that v is a power, or $x = y = z = f(v)$. In the last case, we get by Lemma 1 that $f(v) = t^j$, for some word t with $|t| \mid |u|$ and positive natural number j. Hence, we have $v = f^{-1}(t^j) = (t')^j$. As $|t'| = |t| \mid |u|$, we get that $u \in t'\{t', f(t')\}^*$. □

5 Conclusion

We end this work with some concluding remarks.

First note that the result of Proposition 6 matches the one existing for antimorphic involutions, see Theorem 3. Thus, an optimality of this bound derives an optimality for the antimorphic involution bound, or vice-versa.

The following three examples show that results similar to the ones presented here cannot be derived for more general anti-/morphisms. In all the following examples f can be considered both as a morphism and as an antimorphism over an alphabet that includes $\{a, b\}$ and some natural number $i \geq 1$.

Example 6. Consider the words

$$u = b^i a^i b^i a^{2i} b^{3i} \text{ and } v = b^i a^i b^i a^{2i} b^i,$$

and a function f with $f(a) = \varepsilon$ and $f(b) = b$. Then $w = (uf(u)^2)^\omega = (vf(v)^4)^\omega$ and one can check that there is no t with $|t| \leq |v|$ such that $u, v \in t\{t, f(t)\}^*$. □

Example 7. Consider the words

$$u = a^i b^i a^{2i} \text{ and } v = a^i b^i a^i,$$

and a function f with $f(a) = f(b) = a$. We have $w = (uf(u)^2)^\omega = (vf(v)^3)^\omega$ and exists no t with $|t| \leq |v|$ such that $w \in t\{t, f(t)\}^*$. □

Finally we consider strictly increasing anti/morphisms.

Example 8. Consider the words

$$u = (ab)^{2i-1} a \text{ and } v = a,$$

and a function f with $f(a) = bab$ and $f(b) = aba$. Then $w = (uf(u))^\omega = (vf(v))^\omega$, but u is not part of $\{v, f(v)\}^*$. □

References

1. Fine, N.J., Wilf, H.S.: Uniqueness theorem for periodic functions. Proceedings of the American Mathematical Society 16, 109–114 (1965)
2. Berstel, J., Boasson, L.: Partial words and a theorem of Fine and Wilf. Theoretical Computer Science 218, 135–141 (1999)
3. Constantinescu, S., Ilie, L.: Generalised Fine and Wilf's theorem for arbitrary number of periods. Theoretical Computer Science 339(1), 49–60 (2005)
4. Constantinescu, S., Ilie, L.: Fine and Wilf's theorem for abelian periods. Bulletin of EATCS 89, 167–170 (2006)
5. Czeizler, E., Kari, L., Seki, S.: On a special class of primitive words. Theoretical Computer Science 411, 617–630 (2010)
6. Lothaire, M.: Combinatorics on Words. Cambridge University Press (1997)
7. Kari, L., Seki, S.: An improved bound for an extension of Fine and Wilf's theorem, and its optimality. Fundamenta Informaticae 191, 215–236 (2010)
8. Shallit, J.: http://www.cs.uwaterloo.ca/~shallit/Talks/wilf3.pdf

Taking It to the Limit: Approximate Reasoning for Markov Processes

Kim Guldstrand Larsen[1], Radu Mardare[1], and Prakash Panangaden[2]

[1] Aalborg University, Denmark
[2] McGill University, Canada

Abstract. We develop a fusion of logical and metrical principles for reasoning about Markov processes. More precisely, we lift metrics from processes to sets of processes satisfying a formula and explore how the satisfaction relation behaves as sequences of processes *and sequences of formulas* approach limits. A key new concept is *dynamically-continuous metric bisimulation* which is a property of (pseudo)metrics. We prove theorems about satisfaction in the limit, robustness theorems as well as giving a topological characterization of various classes of formulas. This work is aimed at providing approximate reasoning principles for Markov processes.

1 Introduction

Probabilistic bisimulation, introduced by Larsen and Skou [15] has become the key concept for reasoning about the equivalence of probabilistic and stochastic systems. Labelled Markov processes are the probabilistic analogs of labelled transition systems; they have state spaces that might be continuous but include discrete state spaces as a special case. The theory of probabilistic bisimulation has been extended to stochastic bisimulation relating processes with continuous-state spaces and continuous distributions [6, 4]. These papers provided a characterization of bisimulation using a negation-free logic.

However, it is also widely realized that probabilistic and stochastic bisimulations are too "exact" for most purposes — they only relate processes with identical behaviours. In applications we need instead to know whether two processes that may differ by a small amount in the real-valued parameters (rates or probabilities) have similar behaviours. These motivated the search for a relaxation of the notion of equivalence of processes.

The metric theory was initiated by Desharnais et al. [8] and greatly developed and explored by van Breugel, Worrell and others [19, 18]. The key idea was to consider a behavioral *pseudometric*, i.e. a variation of the concept of metric for processes where pairs of distinct processes are at distance 0 whenever the processes are bisimilar.

Though behavioural pseudometrics were defined, approximate *reasoning principles* as such did not develop. The present work is a step in that direction. We lift the metric between processes to a metric between logical formulas by standard techniques, using the *Hausdorff metric*; but then we break new ground by exploring the relationship between convergence of processes and of formulas. We thus lay the groundwork for a notion of *approximate reasoning* not by getting rid of the logic but by fusing metric and

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 681–692, 2012.

logical principles. The completeness theorems of [4, 5] are a powerful impetus for the present paper.

Consider the sequence of stochastic processes represented in Figure 1. The process m has only one state and one self-transition at rate 5; similarly, for each $k \in \mathbb{N}$, the process m_k has one state and one transition at rate $4.\underbrace{9..9}_{k}$. Using a behavioral pseudometric, we expect to prove that the sequence $(m_k)_{k\in\mathbb{N}}$ of processes converges to m. We often meet such problems in practice where m is a natural process that we need to analyze, while m_k are increasingly accurate models of m. If, in addition, we have a convergent sequence of logical formulas ϕ_k with limit ϕ such that $m_k \models \phi_k$ for each k, we want to understand whether we can infer $m \models \phi$.

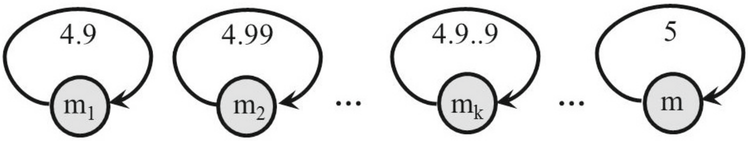


Fig. 1. A sequence of convergent stochastic processes and their limit

In order to address such problems, we identify a general metrical notion that we call *dynamical continuity*. It characterizes the behavioral pseudometrics for which a sequence of processes as the one in our example is convergent; and it allows us to relate the convergence in formulas with convergence of the processes.

Using this concept we can address the above mentioned problem and prove that in general we do not have, at the limit, $m \models \phi$. For the probabilistic case $m \models \phi$ only if ϕ is a *positive* formula. Positive formulas will be defined in the paper; they are restricted, but they suffice for the modal characterization of probabilistic bisimulation. For the stochastic case we have to restrict the set of formulas slightly more, remaining however within a set of formulas that characterize bisimulation. In either case, even if $m \not\models \phi$, there exists a sequence of processes $(n_k)_{k\in\mathbb{N}}$ such that $\lim_{k\to\infty} n_k = m$ and $n_k \models \phi$ for each $k \in \mathbb{N}$. So this gives a handle on constructing approximations satisfying prescribed conditions. Along the way we give topological characterizations of various classes of formulas as defining open, closed, G_δ or F_σ sets[1]. All these results hold whenever one has a dynamically-continuous metric bisimulation, as it is the case with the behavioral pseudometrics introduced in [8, 19, 18].

The Relevance of This Work. We prove that the process of extrapolating properties from arbitrary accurate approximations of a system to the system itself – a method widely accepted as valid and used in applications – is not always consistent. Often one constructs better and better approximations of a system, proves properties of these approximations and extrapolates the results to the original system. But can we indeed

[1] In topology, a G_δ set is a countable intersection of open sets and a F_σ set in a countable union of closed sets.

be sure that if, for instance, the approximants show oscillatory behaviours [3] then the original system also oscillates? The mathematical framework developed in this paper allows us to address such a question and to prove that the answer is **no**, in general: there may exist sequences of arbitrarily accurate approximations of a system showing properties that are not preserved to the limit; and this already happens for fragments of modal probabilistic and stochastic logics, less expressive than CSL or pCTL. We prove that the *preservation to the limit* depends on the logical structure of the property; "negative information" and "approximations from above", for instance, are obstructions to this kind of limiting argument. Moreover, different logics behave differently to the limit. In this paper we show that there is a considerable difference between probabilistic and stochastic logical properties.

2 Preliminaries

In this section we introduce notation and establish terminology. We assume that the basic terminology of topology and measure theory is familiar to the reader. In appendix we collect some basic definitions and the proofs of the major results.

Sets and Measurability. If (M, Σ) is a measurable space with σ-algebra $\Sigma \subseteq 2^M$, we use $\Delta(M, \Sigma)$ to denote the set of measures $\mu : \Sigma \to \mathbb{R}^+$ on (M, Σ) and $\Pi(M, \Sigma)$ to denote the set of probability measures $\mu : \Sigma \to [0, 1]$ on (M, Σ).

We organize $\Delta(M, \Sigma)$ and $\Pi(M, \Sigma)$ as measurable spaces: for arbitrary $S \in \Sigma$ and $r > 0$, let $\Theta = \{\mu \in \Delta(M, \Sigma) : \mu(S) \leq r\}$ and $\Omega = \{\mu \in \Pi(M, \Sigma) : \mu(S) \leq r\}$; let $\overline{\Theta}$ and $\overline{\Omega}$ be the σ-algebras generated by Θ and Ω on $\Delta(M, \Sigma)$ and $\Pi(M, \Sigma)$ respectively.

Given two measurable spaces (M, Σ) and (N, Σ'), we use $[\![M \to N]\!]$ to denote the class of measurable mappings from (M, Σ) to (N, Σ').

Given a relation $\Re \subseteq M \times M$, the set $N \subseteq M$ is $\Re$-closed iff $\{m \in M \mid \exists n \in N, (n, m) \in \Re\} \subseteq N$. If (M, Σ) is a measurable space, we denote $\Sigma(\Re) = \{S \in \Sigma \mid S \text{ is } \Re\text{-closed}\}$.

Distances. Let M be a set. A function $d : M \times M \to \mathbb{R}^+$ is a *pseudometric* on M if it satisfies, for arbitrary $x, y, z \in M$, the following axioms.

$$(1): d(x, x) = 0 \quad (2): d(x, y) \leq d(x, z) + d(z, y) \quad (3): d(x, y) = d(y, x).$$

If d is a pseudometric, (M, d) is a *pseudometric space*.

Given a pseudometric space (M, d), we define the following distances for arbitrary $a \in M$ and $A, B \subseteq M$ with $A \neq \emptyset \neq B$.

$$(1): d^h(a, B) = \inf_{b \in B} d(a, b), \qquad (2): d^{H/2}(A, B) = \sup_{a \in A} \inf_{b \in B} d(a, b),$$

$$(3): d^H(A, B) = max\{\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b)\}.$$

We call d^H the *Hausdorff pseudometric* (associated to d).

Lemma 1. *If (M, d) is a pseudometric space and $\overline{X}$ is the closure of $X \subseteq M$ in the open ball topology $\mathcal{T}_d$, then for arbitrary $A, B \subseteq M$,*
(1): $d^H(A, B) = 0$ iff $\overline{A} = \overline{B}$, (2): $d^H(A, B) = d^H(\overline{A}, B) = d^H(A, \overline{B}) = d^H(\overline{A}, \overline{B})$.

In what follows, we consider for pseudometric spaces (M, d) the following notions of convergence in the open ball topologies $\mathcal{T}_d$ and $\mathcal{T}_{d^H}$ respectively[2]:

[2] A pseudometric space is not a Hausdorff space and consequently the limits are not unique.

– For an arbitrary sequence $(m_k)_{k\in\mathbb{N}}$ of elements of M and an arbitrary $m \in M$, we write $m \in \lim_{k\to\infty} m_k$ (or $\lim_{k\to\infty} m_k \ni m$) to denote that $\lim_{k\to\infty} d(m_k, m) = 0$.
– For an arbitrary sequence $(S_k)_{k\in\mathbb{N}}$ of subsets of M and an arbitrary set $S \subseteq M$, we write $S \in \lim_{k\to\infty} S_k$ (or $\lim_{k\to\infty} S_k \ni S$) to denote that $\lim_{k\to\infty} d^H(S_k, S) = 0$.

Lemma 2. *Let (M, d) be a pseudometric space and $(B_i)_{i\in\mathbb{N}}$ a decreasing sequence of compact subsets of M in the topology $\mathcal{T}_d$. If $\lim_{k\to\infty} B_k \ni A$, then $d^H(A, \bigcap_{i\in\mathbb{N}} B_i) = 0$.*

Lemma 3. *Let (M, d) be a pseudometric space and $(m_k)_{k\in\mathbb{N}} \subseteq M$, $(S_k)_{k\in\mathbb{N}} \subseteq 2^M$ convergent sequences with $m \in \lim_{k\to\infty} m_k$ and $S \in \lim_{k\to\infty} S_k$. If $m_k \in S_k$ for each $k \in \mathbb{N}$, then $d^h(m, S) = 0$. In particular, if S is closed, then $m \in S$.*

3 The Pseudometric Spaces of Processes

In this section we introduce two classes of processes, *discrete-time Markov processes* (DMPs), which are similar to the ones studied in [17, 6, 10]; and *continuous-time Markov processes* (CMPs) [4], which are models of stochastic systems with continuous time transitions. We emphasize that the terms "discrete" and "continuous" refer to time and not to the state space. In this paper we use for both classes the definitions proposed in [4, 5], which exploits an equivalence between the definitions of Harsanyi type spaces [16] and a coalgebraic view of labelled Markov processes [12]. However, with respect to [4, 5] or to [17, 6, 10], we do not consider action labels. The subtle issues all involve convergence and other analytical aspects; the labels can easily be added without changing any of these aspects of the theory.

Definition 1 (Processes). *Let (M, Σ) be an* analytic space*, where Σ is its Borel algebra.*

- *A* discrete Markov kernel *(DMK) is a tuple $\mathcal{M} = (M, \Sigma, \theta)$, where $\theta \in [\![M \to \Pi(M, \Sigma)]\!]$; if $m \in M$, $(\mathcal{M}, m)$ is a* discrete Markov process.
- *A* continuous Markov kernel *(CMK) is a tuple $\mathcal{M} = (M, \Sigma, \theta)$, where $\theta \in [\![M \to \Delta(M, \Sigma)]\!]$; if $m \in M$, $(\mathcal{M}, m)$ is a* continuous Markov process.

For both types of processes, M is called the support set *of $\mathcal{M}$ denoted by $supp(\mathcal{M})$.*

If m is the current state of a DMP and N is a measurable set of states, the transition function $\theta(m)$ is a probability measure on the state space and $\theta(m)(N) \in [0, 1]$ represents the *probability* of a transition from m to an arbitrary state $n \in N$.

Similarly, if m is the current state of a CMP and N is a measurable set of states, the transition function $\theta(m)$ is a measure on the state space and $\theta(m)(N) \in \mathbb{R}^+$ represents the *rate* of an exponentially distributed random variable that characterizes the duration of a transition from m to an arbitrary state $n \in N$. Indeterminacy in such systems is resolved by races between events executing at different probabilities/rates.

Notice that, in both cases, θ is a measurable mapping between the space of processes and the space of (probabilistic/stochastic) measures. These requirements are equivalent to the conditions on the corresponding two-variable *probabilistic/rate function* used in [17, 6, 10] to define labelled Markov processes and in [9] to define continuous Markov processes (for the proof see, Proposition 2.9 [10]).

The definitions of bisimulation for DMPs and CMPs follow the line of the Larsen-Skou definition of probabilistic bisimulation [15].

Definition 2 (Bisimulation). *Given the DMK (CMK) $\mathcal{M} = (M, \Sigma, \theta)$, a* bisimulation relation *on $\mathcal{M}$ is a relation $\Re \subseteq M \times M$ such that whenever $(m, n) \in \Re$, for any $C \in \Sigma(\Re)$, $\theta(m)(C) = \theta(n)(C)$. Two processes $(\mathcal{M}, m)$ and $(\mathcal{M}, n)$ are* bisimilar*, written $m \sim_{\mathcal{M}} n$, if they are related by a bisimulation relation.*

The bisimulation relation between processes with different Markov kernels is defined by taking the disjoint union of the two [17, 6, 4, 5]. For this reason, in what follows we use $\sim$ without extra indices to denote the largest bisimulation relation. We call the largest bisimulation of DMPs *probabilistic bisimulation* and the largest bisimulation of CMPs *stochastic bisimulation*.

As we have already underlined in the introduction, the concept of bisimulation for probabilistic or stochastic processes is very strict. We can however relax it by introducing a behavioral pseudometric [8, 17] which, formally, is a distances between processes that measure their similarity in terms of quantitative behaviour: the kernel of a behavioral pseudometric is a bisimulation. Moreover, we expect that a behavioral pseudometric can prove that a sequence of processes as $(m_k)_{k \in \mathbb{N}}$ represented in Figure 1 is convergent to m. Hereafter in this section we identify a sufficient condition satisfied by any behavioral pseudometric that can prove such a convergence.

Before proceeding with the definition, recall that the convergences are in the corresponding open ball topologies, as defined in the preliminary section. In addition, we define the *kernel* of d as being the set $ker(d) = \{(m, n) \in M \times M \mid d(m, n) = 0\}$.

Definition 3 (Dynamically-continuous metric bisimulation). *Given the DMK (CMK) $\mathcal{M} = (M, \Sigma, \theta)$, a pseudometric $d : \mathcal{M} \times \mathcal{M} \to \mathbb{R}^+$ is*

- *a* metric bisimulation *if $ker(d) = \sim$*
- dynamically-continuous *if whenever $(m_k)_{k \in \mathbb{N}} \in \mathcal{M}$ with $\lim\limits_{k \to \infty} m_k \ni m$ (in $\mathcal{T}_d$), for any $S \in \Sigma(\sim)$ there exists a decreasing sequence $(S_k)_{k \in \mathbb{N}} \subseteq \Sigma(\sim)$ of compact sets in the topology $\mathcal{T}_d$ such that $\lim\limits_{k \to \infty} S_k \ni S$ (in $\mathcal{T}_{d^H}$) and $\lim\limits_{k \to \infty} \theta(m_k)(S_k) = \theta(m)(S)$.*

All the behavioral pseudometrics defined in [8, 19, 18] are dynamically-continuous metric bisimulations. Notice also the coinductive nature of this definition, which is reminiscent of the general definition of bisimulation.

Example 1. Let us now convince ourselves that any behavioral pseudometric that can prove the convergence in Figure 1 is indeed a dynamically-continuous metric bisimulation. Formally, in Figure 1 we have represented the CMK $\mathcal{M} = (M, \Sigma, \theta)$ where $M = \{m, m_1, ..m_k, ..\}$, $\Sigma = 2^M$, $\theta(m_k)(\{m_k\}) = 4.\underbrace{9..9}_{k}$ for $k \in \mathbb{N}$ and $\theta(m)(\{m\}) = 5$.

If in the open ball topology we can prove that $\lim\limits_{k \to \infty} m_k \ni m$, then for each $k \in \mathbb{N}$, $S_k = \{m, m_k, m_{k+1}, ..\}$ is a compact set – since it is closed and bounded in a complete pseudometric space; $S_k \supseteq S_{k+1}$ and $\lim\limits_{k \to \infty} S_k \ni \{m\}$. Because $\theta(m_k)$ is a measure, $\theta(m_k)(S_k) = \theta(m_k)(\{m_k\}) + \theta(m_k)(S_{k+1})$. But $\theta(m_k)(S_{k+1}) = 0$, hence $\theta(m_k)(S_k) = \theta(m_k)(\{m_k\})$. This implies that $\lim\limits_{k \to \infty} \theta(m_k)(S_k) = \lim\limits_{k \to \infty} \theta(m_k)(\{m_k\}) = \theta(m)(\{m\})$ and verifies the second condition of the previous definition. All these arguments motivate our choice for the definition of dynamically-continuous metric bisimulation. □

4 Markovian Logics

In this section we present two logics for Markovian processes: the *discrete Markovian Logic* (DML) for semantics based on DMPs, similar to the logics introduced in the literature, for example in [2, 13, 15]; and the *continuous Markovian logic* (CML) for semantics based on CMPs [4]. In addition to the boolean operators, these logics are endowed with *probabilistic/stochastic modal operators* that approximate the probabilities/rates of transitions. For $r \in \mathbb{Q}^+$, $L_r\phi$ characterizes $(\mathcal{M}, m)$ whenever the probability/rate of the transition from m to the class of states satisfying ϕ is *at least* r; symmetrically, $M_r\phi$ is satisfied when this probability/rate is *at most* r.

Definition 4 (Syntax). *The formulas of $\mathcal{L}$ are introduced by the following grammar.*

$$\mathcal{L}: \quad \phi := \top \mid \neg\phi \mid \phi \wedge \phi \mid L_r\phi \mid M_r\phi, \qquad r \in \mathbb{Q}_+.$$

We isolate the fragment $\mathcal{L}[0,1] \subseteq \mathcal{L}$ defined only for $r \in [0,1] \cap \mathbb{Q}^+$. $\mathcal{L}$ contains the well-formed formulas of CML, $\mathcal{L}[0,1]$ contains the well-formed formulas of DML. As usual, both logics use all the boolean operators including $\bot = \neg\top$.

The major difference between the two logics is reflected in their semantics. The semantics of DML is defined for DMPs, while the semantics of CML is defined in terms of CMPs. The satisfiability relations is similar for the two logics. It assumes a fixed structure $\mathcal{M} = (M, \Sigma, \theta)$ that represents a DMK when it refers to DML, and a CMK when it refers to CML; and $m \in M$ is an arbitrary process.

$$\begin{array}{l} m \models \top \text{ always}, \\ m \models \neg\phi \text{ iff it is not the case that } m \models \phi, \\ m \models \phi \wedge \psi \text{ iff } m \models \phi \text{ and } m \models \psi, \\ m \models L_r\phi \text{ iff } \theta(m)([\![\phi]\!]) \geq r, \\ m \models M_r\phi \text{ iff } \theta(m)([\![\phi]\!]) \leq r, \end{array}$$

where $[\![\phi]\!] = \{m \in M \mid m \models \phi\}$.

When it is not the case that $m \models \phi$, we write $m \not\models \phi$.

The semantics of $L_r\phi$ and $M_r\phi$ are well defined only if $[\![\phi]\!]$ is measurable. This is guaranteed by the fact that θ is a measurable mapping – for the proof see [4].

In spite of their apparent similarities, the two logics are very different at the provability level. Below we present a complete axiomatization for DML in Table 1 [20] and a complete axiomatization for CML in Table 2 [4]. The key differences between the two logics consists of the relation between $L_r\phi$ and $M_r\phi$. In the discrete logic the two are related by De Morgan dualities, stating that the probability of a transition to a state satisfying ϕ depends of the probability of a transition to some state satisfying $\neg\phi$: $\vdash L_r\phi \leftrightarrow M_{1-r}\neg\phi$ and $\vdash M_r\phi \leftrightarrow L_{1-r}\neg\phi$. In the continuous case, the two modal operators are independent [4]. We will see in the following sections that this difference is deeply reflected in the topologies of the two spaces of formulas.

There exist strong relations between logical equivalence and bisimulation both for the probabilistic and for the stochastic cases. In [9, 17] it was shown that the logical equivalence induced by $\mathcal{L}[0,1]$ on the class of DMPs coincides with probabilistic bisimulation. A similar result holds for CML, [4].

Table 1. The axiomatic system of DML

(A1): $\vdash L_0\phi$
(A2): $\vdash L_r\top$
(A3): $\vdash L_r\phi \leftrightarrow M_{1-r}\neg\phi$
(A4): $\vdash L_r\phi \rightarrow \neg L_s\neg\phi,\ r+s>1$
(A5): $\vdash L_r(\phi \wedge \psi) \wedge L_s(\phi \wedge \neg\psi) \rightarrow L_{r+s}\phi,\ r+s \leq 1$
(A6): $\vdash \neg L_r(\phi \wedge \psi) \wedge \neg L_s(\phi \wedge \neg\psi) \rightarrow \neg L_{r+s}\phi,\ r+s \leq 1$
(R1): If $\vdash \phi \rightarrow \psi$ then $\vdash L_r\phi \rightarrow L_r\psi$
(R2): $\{\neg M_r\psi \mid r<s\} \vdash L_s\psi$

Table 2. The axiomatic system of CML

(B1): $\vdash L_0\phi$
(B2): $\vdash L_{r+s}\phi \rightarrow \neg M_r\phi,\ s>0$
(B3): $\vdash \neg L_r\phi \rightarrow M_r\phi$
(B4): $\vdash \neg L_r(\phi \wedge \psi) \wedge \neg L_s(\phi \wedge \neg\psi) \rightarrow \neg L_{r+s}\phi$
(B5): $\vdash \neg M_r(\phi \wedge \psi) \wedge \neg M_s(\phi \wedge \neg\psi) \rightarrow \neg M_{r+s}\phi$
(S1): If $\vdash \phi \rightarrow \psi$ then $\vdash L_r\phi \rightarrow L_r\psi$
(S2): $\{L_r\psi \mid r<s\} \vdash L_s\psi$
(S3): $\{M_r\psi \mid r>s\} \vdash M_s\psi$
(S4): $\{L_r\psi \mid r>s\} \vdash \bot$

5 The Topological Space of Logical Formulas

Since a dynamically-continuous metric bisimulation is a relaxation of the bisimulation relation, in what follows we try to identify similar logical characterization results for dynamically-continuous metric bisimulation. In order to do this, we organize the space of the logical formulas as a pseudometric space, by identifying a logical formula with the set of its models and using the Hausdorff distance.

Formally, assume that the space $\mathcal{M}$ of the continuous (or discrete) Markov kernel is a pseudometric space defined by $d : \mathcal{M} \times \mathcal{M} \rightarrow \mathbb{R}^+$. The Hausdorff pseudometric d^H associated to d is a distance between the sets $[\![\phi]\!]$ of models, for arbitrary $\phi \in \mathcal{L}$ (or $\phi \in \mathcal{L}[0,1]$ respectively). Consequently, we can define, for arbitrary $\phi, \psi \in \mathcal{L}$ (or $\phi, \psi \in \mathcal{L}[0,1]$ respectively), a distance δ by $\delta(\phi, \psi) = d^H([\![\phi]\!], [\![\psi]\!])$.

Proposition 1. *$(\mathcal{L}, \delta)$ and $(\mathcal{L}[0,1], \delta)$ are pseudometric spaces.*

5.1 The Topology of Discrete Markovian Logic

In this subsection we concentrate on the discrete Markovian logic. Let $\mathcal{M} = (M, \Sigma, \theta)$ be the universal DMP organized as a pseudometric space by the behavioural pseudometric d. Let $(\mathcal{L}[0,1], \delta)$ be the pseudometric space of logical formulas.

To understand deeper the relation between the induced topologies, in what follows we isolate the following fragments of $\mathcal{L}[0,1]$.

$$\mathcal{L}[0,1]^+ : \quad f := \top \mid f \vee f \mid f \wedge f \mid L_r\phi \mid M_r\phi, \quad \phi \in \mathcal{L}[0,1],$$
$$\mathcal{L}[0,1]^- = \{\neg f \mid f \in \mathcal{L}[0,1]^+\}.$$

As in the preliminaries, we use $\mathcal{T}_d$ to denote the open ball topology, $\overline{X}$ and X^{int} to denote the closure and the interior of $X \subseteq \mathcal{M}$, respectively.

Proposition 2. *Let d be a dynamically-continuous metric bisimulation on $\mathcal{M}$.*
1. If $\phi \in \mathcal{L}[0,1]^+$, then $[\![\phi]\!]$ is a closed set in the topology $\mathcal{T}_d$.
2. If $\phi \in \mathcal{L}[0,1]^-$, then $[\![\phi]\!]$ is an open set in the topology $\mathcal{T}_d$.
3. $\overline{[\![\neg M_r\phi]\!]} = [\![L_r\phi]\!]$ and $\overline{[\![\neg L_r\phi]\!]} = [\![M_r\phi]\!]$.
4. $[\![M_r\phi]\!]^{int} = [\![\neg L_r\phi]\!]$ and $[\![L_r\phi]\!]^{int} = [\![\neg M_r\phi]\!]$.

At this point we want to understand more about the kernel of δ and its relation to provability. Since the axiomatic system of DML is complete, the next theorem follows from the definition of Hausdorff distance.

Theorem 1. *If $\phi, \psi \in \mathcal{L}[0,1]$ and $\vdash \phi \leftrightarrow \psi$, then $\delta(\phi,\psi) = 0$.*

The next results and the example will show that actually the reverse of Theorem 1 is not true: not all the formulas at distance zero are logically equivalent.

Theorem 2. *Let d be a dynamically-continuous metric bisimulation and $\phi, \psi \in \mathcal{L}[0,1]$ such that $\delta(\phi,\psi) = 0$.*
1. If $\phi \in \mathcal{L}[0,1]^+$, then $\vdash \psi \to \phi$.
2. If $\phi, \psi \in \mathcal{L}[0,1]^+$, then $[\delta(\phi,\psi) = 0$ iff $\vdash \phi \leftrightarrow \psi]$.

Proof. 1. $\delta(\phi,\psi) = 0$ is equivalent to $d^H([\![\phi]\!],[\![\psi]\!]) = 0$, which is equivalent, as stated in Lemma 1, to $\overline{[\![\phi]\!]} = \overline{[\![\psi]\!]}$. If $\phi \in \mathcal{L}[0,1]^+$, by Proposition 2, $[\![\phi]\!]$ is closed. Hence, $\overline{[\![\psi]\!]} = [\![\phi]\!]$. This implies that $[\![\psi]\!] \subseteq [\![\phi]\!]$, i.e., $\models \psi \to \phi$, which is equivalent to $\vdash \psi \to \phi$. □

Example 2. There exist logical formulas that are at distance zero without being logically equivalent:

$$\delta(L_r\phi, \neg M_r\phi) = 0, \quad \vdash \neg M_r\phi \to L_r\phi \quad \text{but} \ \nvdash L_r\phi \to \neg M_r\phi.$$

To prove these, observe that for any model $m \in \mathcal{M}$, if $m \models \neg M_r\phi$, then $m \models L_r\phi$. This guarantees that the closure of $[\![\neg M_r\phi]\!]$ is included in $[\![L_r\phi]\!]$. Observe that if $\theta(m)([\![\phi]\!]) = r$, then $m \models L_r\phi$, but $m \not\models \neg M_r\phi$.

Suppose that there exists a model $m \in \mathcal{M}$ such that $m \in [\![L_r\phi]\!]$ and $d^h(m, [\![\neg M_r\phi]\!]) > 0$. Then, $\theta(m)([\![\phi]\!]) \geq r$ and $\theta(m)([\![\phi]\!]) < r$ - impossible. Hence the closure of $[\![\neg M_r\phi]\!]$ coincides with $[\![L_r\phi]\!]$ and this proves that $\delta(L_r\phi, \neg M_r\phi) = 0$. □

The next theorem states that whenever $(m_k)_{k\in\mathbb{N}}$ is a sequence of increasingly accurate approximations of m, if $m_k \models \phi_k$ for each k, we cannot guarantee that m satisfies the limit ϕ of $(\phi_k)_{k\in\mathbb{N}}$; but, there exists a sequence of approximations of m satisfying ϕ.

Theorem 3. *If d is a dynamically-continuous metric bisimulation and $(\phi_k)_{k\in\mathbb{N}} \subseteq \mathcal{L}[0,1]$, $(m_k)_{k\in\mathbb{N}} \subseteq \mathcal{M}$ are two convergent sequences such that $\lim\limits_{k\to\infty} \phi_k \ni \phi$, $\lim\limits_{k\to\infty} m_k \ni m$ and for each $k \in \mathbb{N}$, $m_k \models \phi_k$, then there exists a convergent sequence $(n_k)_{k\in\mathbb{N}} \subseteq \mathcal{M}$ such that $\lim\limits_{k\to\infty} n_k \ni m$ and $n_k \models \phi$ for each $k \in \mathbb{N}$.*

Proof. If we apply Lemma 3 for $S_k = [\![\phi_k]\!]$ and $S = [\![\phi]\!]$, we obtain that $d^h(m, [\![\phi]\!]) = 0$ which implies that there exists a sequence $(n_k)_{k\in\mathbb{N}} \subseteq [\![\phi]\!]$ such that $\lim\limits_{k\to\infty} n_k = m$. □

There exist, however, properties that can be ”taken to the limit”.

Theorem 4. *Let d be a dynamically-continuous metric bisimulation and $(\phi_k)_{k\in\mathbb{N}} \subseteq \mathcal{L}[0,1]$, $(m_k)_{k\in\mathbb{N}} \subseteq \mathcal{M}$ two convergent sequences such that $\lim\limits_{k\to\infty} \phi_k \ni \phi$, $\lim\limits_{k\to\infty} m_k \ni m$ and $m_k \models \phi_k$ for each $k \in \mathbb{N}$. If $\phi \in \mathcal{L}[0,1]^+$, then $m \models \phi$.*

Proof. As in Theorem 3, $d^h(m, [\![\phi]\!]) = 0$. Since $\phi \in \mathcal{L}[0,1]^+$, Proposition 2 guarantees that $[\![\phi]\!]$ is closed and using the second part of Lemma 3 we obtain $m \in [\![\phi]\!]$. □

5.2 The Topology of the Continuous Logic

In this subsection we investigate similar problems for the case of CMPs and continuous Markovian logic. Hereafter, let $\mathcal{M}$ be the universal CMP organized as a pseudometric space by the behavioural pseudometric d. Let $(\mathcal{L}, \delta)$ be the pseudometric space of logical formulas.

Lemma 4. *For arbitrary $\phi \in \mathcal{L}$,*

1. $\overline{[\![\neg M_r\phi]\!]} = [\![L_r\phi]\!]$,
2. $[\![M_r\phi]\!]^{int} = [\![\neg L_r\phi]\!]$,
3. $[\![M_r\phi]\!] = \bigcap\limits_{k\in\mathbb{N}} [\![\neg L_{r+\frac{1}{k}}\phi]\!]$
4. $[\![L_r\phi]\!] = \bigcap\limits_{k\in\mathbb{N}} [\![\neg M_{r-\frac{1}{k}}\phi]\!]$.

In the following examples we show that, unlike in the probabilistic case, $[\![M_r\psi]\!]$ is sometimes neither open nor closed in $\mathcal{T}_d$.

Example 3. We return to the stochastic system described in Example 1 and represented in Figure 1. Notice that, for each $k \in \mathbb{N}$, $m_k \models M_0L_5\top$ meaning that each m_k cannot do a transition to a state (which is equivalent with ”it does it at rate 0”) where from it is possible to do a transition at rate at least 5. But at the limit, $m \models \neg M_0L_5\top$ since $\theta(m)([\![L_5\top]\!]) = 5 > 0$. Consequently, $[\![M_0L_5\top]\!]$ is not closed in $\mathcal{T}_d$, since we found a sequence of processes from $[\![M_0L_5\top]\!]$ with a limit outside $[\![M_0L_5\top]\!]$.

To prove that sometimes $[\![M_r\psi]\!]$ is not open either, consider the same processes as before only that for each $k \in \mathbb{N}$, $\theta(m_k)(\{m_k\}) = r_k$, where $(r_k)_{k\in\mathbb{N}} \in \mathbb{Q}^+$ is a strictly decreasing sequence with limit 5. In this case, for each $k \in \mathbb{N}$, $m_k \models \neg M_5\top$, since $\theta(m_k)(\{m_k\}) > 5$. However, to the limit we have $m \models M_5\top$ proving that $[\![\neg M_5\top]\!]$ is not closed in $\mathcal{T}_d$, hence, $[\![M_5\top]\!]$ is not open. □

To understand this topology more deeply, we isolate the following fragments of $\mathcal{L}$.

$$\begin{array}{llr}
\mathcal{L}^+ : & f := \top \mid f\wedge f \mid f\vee f \mid L_r\phi \mid M_r\phi, & \mathcal{L}^- = \{\neg f \mid f \in \mathcal{L}^+\},\\
\mathcal{L}_0 : & f := \top \mid f\wedge f \mid \neg f \mid L_r\phi, & \\
\mathcal{L}_0^+ : & f := \top \mid f\wedge f \mid f\vee f \mid L_r\phi, & \mathcal{L}_0^- = \{\neg f \mid f \in \mathcal{L}_0^+\},
\end{array}$$

where in the previous definitions $\phi \in \mathcal{L}[0,1]^+$.

The next lemma marks essential differences between the topology of DML formulas and the topology of CML formulas. Recall that, in topology, a G_δ set is a countable intersection of open sets and a F_σ set is a countable union of closed sets.

Theorem 5. *If d is a dynamically-continuous metric bisimulation, then*
1. If $\phi \in \mathcal{L}_0^+$, then $[\![\phi]\!]$ is a closed set in the topology $\mathcal{T}_d$.
2. If $\phi \in \mathcal{L}_0^-$, then $[\![\phi]\!]$ is an open set in the topology $\mathcal{T}_d$.
3. If $\phi \in \mathcal{L}^+$, then $[\![\phi]\!]$ is a G_δ set in the topology $\mathcal{T}_d$.
4. If $\phi \in \mathcal{L}^-$, then $[\![\phi]\!]$ is a F_σ set in the topology $\mathcal{T}_d$.

As for DML, logical equivalence is a subset of the kernel of δ.

Theorem 6. *If $\vdash \phi \leftrightarrow \psi$, then $\delta(\phi, \psi) = 0$.*

However, Theorem 2 does not hold for CML. Instead we have the following weaker result relying on the fact that $[\![\phi]\!]$ is closed whenever $\phi \in \mathcal{L}_0^+$.

Theorem 7. *Let d be a dynamically-continuous metric bisimulation and $\phi, \psi \in \mathcal{L}$ such that $\delta(\phi, \psi) = 0$.*
1. If $\phi \in \mathcal{L}_0^+$, then $\vdash \psi \rightarrow \phi$.
2. If $\phi, \psi \in \mathcal{L}_0^+$, then $[\delta(\phi, \psi) = 0$ iff $\vdash \phi \leftrightarrow \psi]$.

A similar result to Theorem 3 holds for CML.

Theorem 8. *If d is a dynamically-continuous metric bisimulation and $(\phi_k)_{k\in\mathbb{N}} \subseteq \mathcal{L}$, $(m_k)_{k\in\mathbb{N}} \subseteq \mathcal{M}$ are two convergent sequences such that $\lim_{k\to\infty} \phi_k \ni \phi$, $\lim_{k\to\infty} m_k \ni m$ and $m_k \models \phi_k$ for each $k \in \mathbb{N}$, then there exists a convergent sequence $(n_k)_{k\in\mathbb{N}} \subseteq \mathcal{M}$ such that $\lim_{k\to\infty} n_k \ni m$ and $n_k \models \phi$ for each $k \in \mathbb{N}$.*

Theorem 4 does not hold for CML. But since $[\![\phi]\!]$ is closed whenever $\phi \in \mathcal{L}_0^+$, we have a weaker version of it that does not involve the operators of type M_r.

Theorem 9. *Let d be a dynamically-continuous metric bisimulation and $(\phi_k)_{k\in\mathbb{N}} \subseteq \mathcal{L}$, $(m_k)_{k\in\mathbb{N}} \subseteq \mathcal{M}$ two convergent sequences such that $m_k \models \phi_k$ for each $k \in \mathbb{N}$, $\lim_{k\to\infty} \phi_k \ni \phi$ and $\lim_{k\to\infty} m_k \ni m$. If $\phi \in \mathcal{L}_0^+$, then $m \models \phi$.*

6 Conclusions

The main contributions of the present paper are the following results:

– The definition of dynamically-continuous metric bisimulation which is the correct distance-based counterpart of the concept of probabilistic/stochastic bisimulation.
– The definition of the topology of logical formulas canonically induced by the behavioural pseudometrics.
– Theorems that establish when parallel sequences of (probabilistic or stochastic) processes and formulas converge to give satisfaction *in the limit*; these results reveal important differences between the probabilistic and stochastic Markovian logics.
–Theorems regarding the relationships between logical formulas being at zero distance and logical equivalence/provability.
– Topological characterization of various classes of formulas.

There are many new things to explore. We currently prepare a coalgebraic presentation of this work that helped us understanding a *metric analogue of Stone duality* for Markov processes. These results are in preparation and are directly inspired by the present work. One topic that we have not understood to our satisfaction is the precise relationship between the kernel of the Hausdorff metric on formulas and provability. We would like to find a definition of the logical distance independent of the semantics and, in this sense, we exploit our previous works on completeness of Markovian logics. Another topic that we are currently explore is the relation with the approximation theory for Markov processes that, we believe, can highly benefit from this work.

Acknowledgments. This research was supported by an NSERC grant and an ONR grant to McGill University. Mardare was also supported by Sapere Aude: DFF-Young Researchers Grant 10-085054 of the Danish Council for Independent Research. Mardare and Larsen were also supported by the VKR Center of Excellence MT-LAB and by the Sino-Danish Basic Research Center IDEA4CPS.

References

[1] Arveson, W.: An Invitation to C^*-Algebra. Springer (1976)

[2] Aumann, R.: Interactive epistemology I: knowledge. International Journal of Game Theory 28, 263–300 (1999)

[3] Ballarini, P., Mardare, R., Mura, I.: Analysing Biochemical Oscillations through Probabilistic Model Checking. In: FBTC 2008. ENTCS, vol. 229(1), pp. 3–19 (2009)

[4] Cardelli, L., Larsen, K.G., Mardare, R.: Continuous Markovian logic - from complete axiomatization to the metric space of formulas. In: CSL, pp. 144–158 (2011)

[5] Cardelli, L., Larsen, K.G., Mardare, R.: Modular Markovian Logic. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 380–391. Springer, Heidelberg (2011)

[6] Desharnais, J., Edalat, A., Panangaden, P.: Bisimulation for labeled Markov processes. I&C 179(2), 163–193 (2002)

[7] Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Approximating labeled Markov processes. I&C 184(1), 160–200 (2003)

[8] Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: A metric for labelled Markov processes. TCS 318(3), 323–354 (2004)

[9] Desharnais, J., Panangaden, P.: Continuous stochastic logic characterizes bisimulation for continuous-time Markov processes. JLAP 56, 99–115 (2003)

[10] Doberkat, E.-E.: Stochastic Relations. Foundations for Markov Transition Systems. Chapman and Hall, New York (2007)

[11] Dudley, R.M.: Real Analysis and Probability. Wadsworth and Brookes/Cole (1989)

[12] de Vink, E., Rutten, J.J.M.M.: Bisimulation for probabilistic transition systems: A coalgebraic approach. TCS 221(1/2), 271–293 (1999)

[13] Fagin, R., Halpern, J.Y.: Reasoning about knowledge and probability. JACM 41(2), 340–367 (1994)

[14] Giacalone, A., Jou, C.-C., Smolka, S.A.: Algebraic Reasoning for Probabilistic Concurrent Systems. In: IFIP WG 2.2/2.3 Working Conference on Programming Concepts and Methods (1990)

[15] Larsen, K.G., Skou, A.: Bisimulation through probablistic testing. Information and Computation 94, 1–28 (1991)

[16] Moss, L.S., Viglizzo, I.D.: Harsanyi type spaces and final coalgebras constructed from satisfied theories. ENTCS 106, 279–295 (2004)

[17] Panangaden, P.: Labelled Markov Processes. Imperial College Press (2009)

[18] van Breugel, F., Mislove, M.W., Ouaknine, J., Worrell, J.B.: An Intrinsic Characterization of Approximate Probabilistic Bisimilarity. In: Gordon, A.D. (ed.) FOSSACS 2003. LNCS, vol. 2620, pp. 200–215. Springer, Heidelberg (2003)

[19] van Breugel, F., Worrell, J.B.: An Algorithm for Quantitative Verification of Probabilistic Transition Systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 336–350. Springer, Heidelberg (2001)

[20] Zhou, C.: A complete deductive system for probability logic with application to Harsanyi type spaces. PhD thesis, Indiana University (2007)

Asymmetric Swap-Equilibrium: A Unifying Equilibrium Concept for Network Creation Games

Matúš Mihalák and Jan Christoph Schlegel

Institute of Theoretical Computer Science, ETH Zurich, Switzerland
matus.mihalak@inf.ethz.ch, jansc@student.ethz.ch

Abstract. We introduce and study the concept of an *asymmetric swap-equilibrium* for network creation games. A graph where every edge is *owned* by one of its endpoints is called to be in *asymmetric swap-equilibrium*, if no vertex v can delete its own edge $\{v, w\}$ and add a new edge $\{v, w'\}$ and thereby decrease the sum of distances from v to all other vertices. This equilibrium concept generalizes and unifies some of the previous equilibrium concepts for network creation games. While the structure and the quality of equilibrium networks is still not fully understood, we provide further (partial) insights for this open problem. As the two main results, we show that (1) every asymmetric swap-equilibrium has at most one (non-trivial) 2-edge-connected component, and (2) we show a logarithmic upper bound on the diameter of an asymmetric swap-equilibrium for the case that the minimum degree of the unique 2-edge-connected component is at least n^{ε}, for $\varepsilon > \frac{4 \lg 3}{\lg n}$. Due to the generalizing property of asymmetric swap equilibria, these results hold for several equilibrium concepts that were previously studied. Along the way, we introduce a node-weighted version of the network creation games, which is of independent interest for further studies of network creation games.

1 Introduction

Many communication networks (such as the Internet) are planned, maintained and built locally by individual entities (such as the autonomous systems). This new phenomenon contrasts with centrally planned and built networks. Network creation games study the quality and structure of communication networks that are created in this non-central way.

The first and arguably the most prominent game-theoretic consideration in this field is the *(original) network creation game* [5]) – a strategic game parameterized by a value $\alpha > 0$ in which the players buy adjacent edges at the price α each, and aim to minimize the cost expressed as the usage cost plus the cost for the bought edges, where a usage cost is the sum of all distances from the respective player. A series of papers [5,1,3,7] studied the *structure* and the *price of anarchy* of (Nash) equilibria in network creation games. The price of anarchy is expressed as the social cost of the worst equilibrium network divided by the

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 693–704, 2012.

social cost of an optimum (centrally-planned) network (where, a social cost is the sum of all individual costs of the players). It is believed that the price of anarchy is constant for all values of α. This has been shown for almost all values of α – with the exception of the range $n^{1-\varepsilon} \leq \alpha \leq 273 \cdot n$ (for $\varepsilon \geq 1/\lg n$). It remains a major open problem to prove/disprove the conjecture. In each of these papers, a completely new proof-technique for giving an upper bound on the price of anarchy has been developed, largely depending on the parameter α. To overcome this rather unnatural "dependency" on α, Alon et al. [2] defined and studied a new equilibrium concept: a graph is called to be in *swap equilibrium*, if no vertex v (a player) can delete an existing edge $\{v, w\}$ and add a new edge $\{v, w'\}$ and thereby decrease the usage cost of v. Swap equilibria indeed do not depend on α, but they do fail to generalize Nash equilibria. The authors conjecture that swap equilibria have at most polylogarithmic diameter. We give a partial affirmation of this.

Another approach to "get rid of" α (although having a different motivation than Alon et al. [2]) has been presented by Ehsani et al. [4] which studied the so called *bounded-budget network creation game* – a variant of the original network creation game in which the players' only goal is to minimize their usage cost (as opposed to minimizing the usage cost plus the creation cost in the original game) given that every player can buy at most a given number of edges. While conceptually different, these two approaches enjoy several similarities, which shall become obvious with our work.

Driven by the desire to answer the aforementioned open problem, and following the original idea of Alon et al. to "get rid of" of α, we define and study the *asymmetric swap-equilibrium*, a natural modification of the swap equilibrium: a graph where every edge is *owned* by one of its endpoints is called to be in *asymmetric swap-equilibrium*, if no vertex v can delete its own edge $\{v, w\}$ and add a new edge $\{v, w'\}$ and thereby decrease the usage cost of v. Asymmetric swap-equilibria have, on top of the inherent properties of swap equilibria (such as that best responses can be calculated efficiently), the interesting property that they generalize swap equilibria and they also generalize Nash equilibria in both the original and the bounded-budget network creation games. Thus, any quality and structural "upper bounds" that one proves for asymmetric swap-equilibrium immediately hold for these equilibrium concepts as well.

As solving the main open problem seems to be difficult (as evidenced by the many papers on the topic that only partially solve it), we are also interested in partial results towards this direction. Besides the quality of asymmetric swap-equilibria, we thus also study their structure, which helps understanding equilibrium graphs. In fact, analyzing the diameter of equilibrium networks is another main open problem.

Definition of the Problem and Related Concepts. For every (undirected) graph G we use the following notation. We denote the vertex set of G by $V(G)$ and its edge set by $E(G)$. For $u, v \in V(G)$ we denote by $d_G(u, v)$ the length of a shortest u-v-path in G. If G is not connected we define $d_G(u, v) := \infty$. We denote the diameter of G by $\mathrm{diam}(G)$ and the radius by $\mathrm{rad}(G)$. We sometimes omit writing G and write simply $d(u, v)$, diam or rad if the underlying graph G is clear from the context. Recall that a *2-edge-connected graph* is a graph of size at least 2 that does not contain a *bridge*, i.e. an edge whose removal makes the graph disconnected, and that a *2-edge-connected component* of a graph G is a maximal 2-edge-connected subgraph of G.

The (original) *network creation game* (as defined by Fabrikant et al. [5]) is a strategic game parameterized by a positive real number α called the *edge price*. The game is played by n players $[n]$ representing nodes in a graph, where a strategy s_i of a player $i \in [n]$ is a set of adjacent edges that it *buys* (or *builds*). The (played) strategies $s = (s_1, s_2, \ldots, s_n)$ of the players naturally define the graph $G(s) = ([n], \bigcup s_i)$. The *creation cost* of player i in the game is $\alpha|s_i|$, i.e., the amount it pays for the edges s_i, and the *usage cost* of player i in the game is the sum of distances from i to all other nodes in the graph $G(s)$. The *cost function* $c_i(s)$ of player i is expressed as the creation cost plus the usage cost. A *Nash equilibrium* (NE for short) of the network creation game are strategies $s = (s_1, \ldots, s_n)$ of the players such that no player can lower her current cost $c_i(s)$ by changing its chosen strategy s_i to a different one. It is easy to see that for every finite α every NE induces a connected graph. We call a graph induced by a Nash equilibrium a *stable graph* or simply, by abusing the notation a bit, a *Nash equilibrium*.

The *bounded-budget network creation game* (as defined and studied by Ehsani et al. [4]) is a network creation game without the parameter α where every player i has a *budget* b_i on the number of edges it can buy. The set of strategies S_i contains only sets of adjacent edges of i of cardinality no more than b_i. The cost function c_i of player i is then just the usage cost of the original network creation game. We will see that several of the results for the bounded-budget network creation game carry over to our model studied in this paper.

The *basic network creation game* (as defined and studied by Alon et al. [2]) is not a strategic game. Rather, it is an equilibrium concept for graphs. Analogously to network creation games, each vertex possesses a cost function (which is the usage cost of the original network creation game). A graph G is called to be in *swap equilibrium* if no vertex $v \in V(G)$ can improve its cost by deleting an adjacent edge $\{v, w\} \in E(G)$ and creating a new adjacent edge $\{v, w'\}$.

In this paper we define the *asymmetric swap-equilibrium* based on the *ownership* of edges. An *ownership* of a graph G is a function $o : E(G) \to V(G)$ that assigns to every edge $\{u, v\} \in E(G)$ either u or v. If $o(\{u, v\}) = u$ we say that u *owns* the edge $\{u, v\}$ and that the edge $\{u, v\}$ is owned by u. Again, every vertex (a player) has a cost – the usage cost of the original network creation game. A graph G with an ownership o is called to be in *asymmetric swap-equilibrium* if no vertex $v \in V(G)$ can improve its cost by deleting its *own* adjacent edge $\{v, w\} \in E(G)$ and creating a new adjacent edge $\{v, w'\}$. Such a modification is called a *swap* (of an edge), and results in a modified graph and modified ownership in that the newly created edge is owned by the vertex v.

Asymmetric swap equilibria generalize these equilibrium concepts in the following sense. Every stable graph $G(s)$ of the (original) network creation game induces an ownership o in which each edge is owned by the player which bought it in the Nash equilibrium s (and observe that in a Nash equilibrium no edge is bought by two players). It is easy to see that for this ownership o, the graph $G(s)$ is in asymmetric swap equilibrium. One can make similar arguments about Nash equilibria of the bounded-budget network creation games. Furthermore, a swap equilibria of the basic network creation game is an asymmetric swap-equilibrium for any ownership o. Thus, we have:

Proposition 1. *Every stable graph of the original network creation game, every stable graph of the bounded-budget network creation game, and every swap equilibrium graph is an asymmetric swap-equilibrium graph.*

Another motivation to study (asymmetric) swap equilibria is that computing a best swap of a player is easy (i.e., polynomial), while computing the best strategy s_i of a player i (given the strategies of all other players) in the original/bounded-budget network creation game is an NP-hard problem [5,4]. We note that this is also true for the basic network creation game if an arbitrary number of swaps is permitted [6].

2 The Structure of Asymmetric Swap-Equilibria

In the following we state and prove the main result of this paper.

Theorem 1. *Every graph in asymmetric swap-equilibrium has at most one* 2*-edge-connected component.*

Proof. Let G be an asymmetric swap-equilibrium and assume for contradiction that there are two 2-edge-connected components $H_1, H_2 \subset G$. Assign every vertex $v \in V(G)$ whose shortest v-H_1-path passes through H_2 to H_2 and every vertex $v \in V$ whose shortest v-H_2-path passes through H_1 to H_1. Denote by $\tilde{H}_1$ the vertices assigned to H_1 and by $\tilde{H}_2$ the vertices assigned to H_2. Without loss of generality $\tilde{H}_1$ is the smallest of the two, i.e., $|\tilde{H}_1| \leq |\tilde{H}_2|$. Let $\{x_1, x_2\} \in E(G)$ with $x_1 \in H_1$ be the first edge in the (unique) shortest H_1-H_2-path in G (and observe that $\{x_1, x_2\}$ is a bridge of G). A schematic illustration of the situation is depicted in Figure 1.

The main idea of the proof is the following. Observe that every vertex u of H_1, with the exception of x_1, decreases its distance to every vertex of $\tilde{H}_2$, if an edge $\{u, x_2\}$ is added to G. We will show that there always exists a vertex u in H_1 that owns an edge $\{u, w\}$ such that the deletion of the edge does not increase the usage cost too much so that this vertex v can swap $\{u, w\}$ for $\{u, x_2\}$ and improve its usage cost – a contradiction with the assumption that the graph is in asymmetric swap-equilibrium. We consider three cases: Either there is a vertex u which owns an edge $\{u, v\} \in E(H_1)$ such that v is closer to x_1 than u, or this is not the case but there is a vertex u which owns an edge $\{u, v\} \in E(H_1)$ such that u and v are at the same distance from x_1, or neither is the case and therefore every edge $\{u, v\} \in E(H_1)$ has both its vertices at different distances

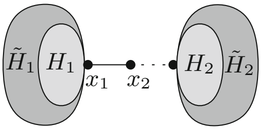


Fig. 1. 2-edge-connected components

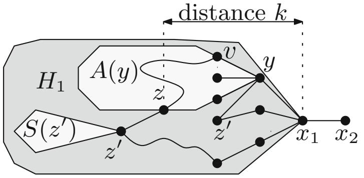


Fig. 2. Case 3 of the proof of Theorem 1

from x_1 and the vertex which owns the edge is closer to x_1 than the other vertex of the edge.

Case 1. In the first case, consider the swap of the edge $\{u, v\}$ with the edge $\{u, x_2\}$. The distance of u to every vertex of $\tilde{H}_2$ decreases by $d(u, x_2) - 1 = d(u, x_1)$ and the distance of u to every vertex of $\tilde{H}_1 \setminus \{u\}$ increases by at most $d(v, x_2) = d(u, x_1)$. The distance of u to any other vertex does not increase. Hence, as $|\tilde{H}_1 \setminus \{u\}| < |\tilde{H}_2|$, u could improve by swapping, a contradiction.

Extra Notation. We will consider all vertices of $\tilde{H}_1$ aligned in *layers* according to the distance from x_1; all vertices at distance k from x_1 will be referred to as *layer* k. We will also call an edge a *layer-edge* if both its endpoints lie in the same layer. Let p be a path connecting two vertices a and b in $\tilde{H}_1$ and $\{u, v\}$ an edge in it, where u is from layer k and v from layer $k + 1$ for some k. We can consider the path as oriented from a to b or vice versa. For a considered orientation, we call $\{u, v\}$ a *forward-edge* in p if u precedes v on p. Similarly, we call $\{u, v\}$ a *backward-edge* in p if v precedes u on p. Thus, "forward"/"backward" reflects on the progression of the path away from x_1.

Case 2. Consider now the second case where a vertex u owns an edge $\{u, v\}$ such that u and v are at the same distance k from x_1, i.e., $\{u, v\}$ is a layer-edge. Consider the swap of edge $\{u, v\}$ with edge $\{u, x_2\}$. Recall that the endpoints of the edge $\{u, v\}$ lie in the same layer, and therefore the distance from x_1 to every vertex of $\tilde{H}_1$ cannot increase after the swap. Let us investigate how much the distances from u to $\tilde{H}_1$ could increase. Let us first consider the simple case when $k = 1$. Then the increase of the distance from u to vertices in $\tilde{H}_1 \setminus \{u\}$ is at most $d(u, x_1) + d(x_1, v) - 1 = 1$. Vertex u decreases its distance to all vertices in $\tilde{H}_2$ by $d(u, x_2) - 1 = 1$, and as $|\tilde{H}_2| > |\tilde{H}_1 \setminus \{u\}|$, u improves its usage cost by the swap, a contradiction. We therefore assume that $k > 1$. The distances from u to every vertex of $\tilde{H}_2$ decrease by $d(u, x_2) - 1 = k$. Let us now consider the increase of the distances from u. In general, the length of a shortest path from u to a vertex $w \in \tilde{H}_1$ could change after the swap, but the length is upper bounded by the length of the u-w-path that uses the new edge $\{u, x_2\}$ and goes via x_1. Obviously, after the swap, u can increase its distance only to vertices w for which a shortest u-w-path p_w uses the edge $\{u, v\}$. We classify the vertices w according to the shortest u-w-path p_w before the swap: (i) path p_w contains, besides $\{u, v\}$, only forward-edges, (ii) p_w contains, besides $\{u, v\}$, exactly one layer-edge and forward-edges, (iii) p_w is none of the first two. Let $S(i)$, $S(ii)$ denote, respectively, the vertices w for which p_w satisfies (i) and (ii). By the swap, the distances of u to vertices in $S(i)$ increase by at most

$1+1+d(x_1,v)-1=k+1$. The distances of u to vertices in $S(ii)$ increase by at most $2+d(x_1,w)-d(u,w) \leq k$. The distances of u to other vertices w increase by at most $k-1$. Comparing the total increase and decrease of the distances from u, the total decrease of the usage cost of vertex u is at least

$$k \cdot |\tilde{H}_2| - (k+1) \cdot |S(i)| - k \cdot |S(ii)| - (k-1) \cdot (|\tilde{H}_1 \setminus \{u\}| - |S(i)| - |S(ii)|) > |\tilde{H}_1| - 2|S(i)| - |S(ii)|.$$

If $|\tilde{H}_1| - 2|S(i)| - |S(ii)| \geq 0$ then, by the above equation, vertex u improves its usage cost by the swap, a contradiction. Assume therefore that $|\tilde{H}_1| - |S(i)| - |S(ii)| < |S(i)|$. But then $S(i)$ contains a lot of vertices and vertex x_1 can swap the incident edge $\{x_1, y\}$ in a shortest x_1-v-path (which x_1 owns as we are not in case 1) with the edge $\{x_1, v\}$ and improve its usage cost: observe that such a swap decreases the distances of x_1 to $S(i)$ by $k-1$, does not increase the distances to $S(ii)$, and increases the distances to $\tilde{H}_1 \setminus \{S(i) \cup S(ii)\}$ by at most $1+d(v,y)-1=k-1$. This is a contradiction.

Further Extra Notation. For the analysis of the third case we introduce the following notation. For every vertex $v \in H_1$ we define $S(v) \subset \tilde{H}_1$ to be the set of vertices $u \in \tilde{H}_1$ such that some (but not necessarily every) shortest u-x_1-path passes through v, and we define $A(v) \subset S(v)$ to be the set of vertices $u \in \tilde{H}_1$ such that all shortest u-x_1-paths pass through v. Note that by definition $v \in A(v)$.

Case 3. Consider now the third case, i.e., the case where every edge $\{u, v\} \in E(H_1)$ has the vertices u, v at different distances from x_1 and the "owner" of the edge is closer to x_1 than the other vertex. Observe that, as there is no layer-edge, and because H_1 is 2-edge-connected, there are at least three layers in H_1 (including the 0-th layer consisting of x_1).

Note that, as H_1 is 2-edge-connected, x_1 has at least 2 neighbors in H_1, and therefore it has a neighbor $y \in V(H_1)$ such that $|A(y)| < |\tilde{H}_1|/2$. Because H_1 is 2-edge-connected, every vertex $v \in A(y)$ has an alternative v-x_1-path in H_1 that does not go via y. Moreover, for every $v \in A(y)$, there is such an alternative v-x_1-path of the following type: starting from v, the first (nonempty) part of the path is using only vertices from $A(y)$, and the second (nonempty) part is using only vertices from $V(H_1) \setminus A(y)$ and is a shortest path to x_1 (see Figure 2). Let us consider the edge $\{z, z'\}$ where such an alternative v-x_1-path (for any vertex v) leaves the first part of the path, i.e., where z is from $A(y)$ and z' is not from $A(y)$ anymore. Obviously, z and z' are from different layers (as we assume there is no layer-edge). Moreover, z has to be closer to x_1 than z' is, as otherwise there would be a shortest path from z to x_1 that does not go via y, a contradiction with the assumption that $z \in A(y)$. Let k denote the distance of z from x_1, i.e., $k = d(x_1, z)$.

First, if $k=1$, i.e., $z=y$, and $\{z, z'\} = \{y, z'\}$, we consider the swap of $\{y, z'\}$ with $\{y, x_2\}$ by vertex y. The distance of y to $\tilde{H}_2$ decreases by $d(y, x_2) - 1 = 1$, the distance of y to $S(z')$ increases by at most $d(y, x_1) + d(x_1, z') - 1 = 2$ (recall that $z' \notin A(y)$ and therefore the swap cannot increase the distance of z' from x_1), and no other distances increase. Therefore, as we assume y cannot improve

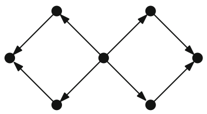


Fig. 3. An example of a graph in asymmetric swap-equilibrium that has two 2-vertex-connected components. The ownership of an edge is depicted by the arrows – the owner is the tail of the respective edge.

by this swap, we have $2|S(z')| \geq |\tilde{H}_2|$ $(\geq |\tilde{H}_1|)$. But then vertex x_1 can improve its usage cost by swapping $\{x_1, y\}$ with $\{x_1, z'\}$: the distance to $S(z')$ decreases by 1, the distance to $A(y)$ increases by 1, and no other distance increases; as $|A(y)| < |\tilde{H}_1|/2$ and $|S(z')| \geq |\tilde{H}_1|/2$, vertex x_1 improves its usage cost. This is a contradiction.

Assume now that $k \geq 2$. Consider the vertex z and the swap of the edge $\{z, z'\}$ with the edge $\{z, x_2\}$. After the swap, vertex z decreases its distance to the vertices in $\tilde{H}_2$ by $d(z, x_2) - 1 = k$. It may increase its distance to the vertices w for which the shortest z-w-path p_w used the edge $\{z, z'\}$. Let us classify such vertices w according to the number of backward-edges in p_w: let $S(i)$ denote the set of vertices $w \neq z$ for which p_w uses only forward-edges (i.e., $S(i)$ is equal to $S(z)$); and let $S(ii)$ denote the set of vertices $w \neq z$ for which p_w uses exactly one backward-edge. Then, after the swap, z increases its distances to vertices in $S(i)$ by at most $1 + 1 + d(x_1, z') - 1 = k + 2$ (z can now get to w via the direct connection to x_2) and it increases its distances to vertices in $S(ii)$ by at most k, and it increases its distances to any other vertex of $\tilde{H}_1 \setminus z$ by at most $k-2$. (This follows because the shortest z-w path p_w to any other vertex w either does not use $\{z, z'\}$, or it uses at least 2 backward-edges). Therefore, the total decrease of the usage cost of z after the swap is at least

$$k \cdot |\tilde{H}_2| - (k+2) \cdot |S(i)| - k \cdot |S(ii)| - (k-2) \cdot (|\tilde{H}_1 \setminus \{z\}| - |S(i)| - |S(ii)|)$$
$$> 2 \cdot |\tilde{H}_1| - 4 \cdot |S(i)| - 2 \cdot |S(ii)|.$$

The graph is in asymmetric swap-equilibrium and therefore z cannot improve by this swap. Thus, $2 \cdot |\tilde{H}_1| - 4 \cdot |S(i)| - 2 \cdot |S(ii)| < 0$, or equivalently, $|\tilde{H}_1| - |S(i)| - |S(ii)| < |S(i)|$. But in this case $S(i)$ is very large and x_1 would benefit from an edge to z': Consider a swap of $\{x_1, y\}$ with $\{x_1, z'\}$; x_1 decreases its distances to $S(i)$ by $d(x_1, z') - 1 = k$; it decreases its distances to $S(ii)$ by $k - 2$ (recall that $k \geq 2$; thus, x_1 cannot increase its distance to $S(ii)$); it increases its distances to any other vertex in $\tilde{H}_1$ by at most $1 + d(z', y) - 1 = k$; thus, x_1 improves by the swap, a contradiction. □

We note that the theorem cannot be made stronger in that there are asymmetric swap-equilibria that have more than one 2-(vertex)-connected components – see Figure 3. Not many constructions of bridge-less (Nash or swap) equilibrium graphs are known (a small cycle or a complete graph are the firm favorites) and they all have diameter ≤ 3 (to the best of our knowledge). Figure 3 gives a simple example of a bridge-less asymmetric swap-equilibrium with diameter 4. Finding less trivial examples is definitely an interesting quest.

Besides the 2-edge-connected component H, an asymmetric swap-equilibrium also contains trees. If H is an empty graph, then the equilibrium is a tree. Ehsani et al. [4] analysed trees in the bounded-budget network creation games. A careful inspection of their proofs shows that their analysis can be taken 1-to-1 to argue about asymmetric swap equilibria, too:

Theorem 2 ([4]). *A tree in asymmetric swap-equilibrium has diameter $\mathcal{O}(\log n)$. Moreover, a complete binary tree where every node owns all adjacent edges to its children is in asymmetric swap-equilibrium.*

2.1 A Vertex-Weighted Version of the Game

An asymmetric swap-equilibrium G thus consists of a nontrivial 2-edge-connected component H with trees attached to the vertices of H. This suggests the following natural variant of the game. Consider a node-weighted graph $\bar{G}$, i.e., a graph with a weight $c(v) \in \mathbb{N}$ for every vertex $v \in V(\bar{G})$. For such a graph, consider the modified usage cost where every distance is "weighted" by the corresponding weight of the vertex: $\sum_{v \in V(\bar{G})} c(v) \cdot d(u,v)$. We further define $c(\bar{G}) := \sum_{v \in \bar{G}} c(v)$ for any graph $\bar{G}$. Instead of studying G, we may study the node-weighted graph $\bar{G}$ where $\bar{G}$ is the 2-edge-connected component H of G, and the weight $c(v)$ of vertex $v \in H$ is the number of vertices in the attached trees. For node-weighted graphs, we are interested in both the swap equilibria and the asymmetric swap-equilibria. Adapting the proofs and results for the non-weighted setting (the missing proofs can be found in [8]), we obtain:

Proposition 2. *Let T be a tree and $c : V(T) \longrightarrow \mathbb{N}$ an arbitrary weight function. If T is in asymmetric swap-equilibrium in the weighted version of the game, then diam$(T) = \mathcal{O}(\log c(T))$.*

Corollary 1. *Let G be a non-tree graph in asymmetric swap-equilibrium for the unweighted version of the game and H be its unique 2-edge-connected component. Then diam$(G) =$ diam$(H) + 2\log n$.*

Proposition 3. *Let T be a tree and $c : V(T) \longrightarrow \mathbb{N}$ a weight function. If T is a swap equilibrium in the weighted version of the game then diam$(T) \leq 2$.*

Corollary 2. *Let G be a non-tree swap equilibrium and H its 2-edge-connected component in the unweighted version of the game. Then diam$(G) \leq$ diam$(H) + 4$.*

3 Diameter of Non-tree Asymmetric Swap-Equilibria

In this section we consider the problem of bounding the diameter of asymmetric swap-equilibria. Ehsani et al. [4] showed that the diameter of any Nash equilibrium in the bounded-budget network creation game is at most $2^{O(\sqrt{\log n})}$. They proved this for a more general concept of equilibrium graphs, which, it turns out, is equivalent to the asymmetric swap-equilibrium. Therefore, they proved:

Theorem 3 ([4]). *The diameter of a graph in asymmetric swap-equilibrium is at most $2^{O(\sqrt{\log n})}$.*

It is believed, however, that the diameter of equilibrium graphs is much smaller. Similarly to the original network creation game, where for various values of α different techniques have been applied to show constant price of anarchy, we believe that studying various classes of asymmetric swap-equilibria can have a similar effect. In the following we focus on a special case where the unique 2-edge-connected component of an equilibrium graph has a large minimum degree. Along the way we present a more general approach that can possibly be applied to more general (asymmetric) swap-equilibria. Let G be a non-tree asymmetric swap-equilibrium on n vertices. By Theorem 1 we know that G has a unique 2-edge-connected component H. We will show that in our special case, the diameter of H is a constant, and hence, by Corollary 1, a $O(\log n)$ upper bound on the diameter of G, or respectively, by Corollary 2, a constant upper bound on the diameter of G if G is a swap equilibrium. This problem can also be seen as a problem to show a constant (with respect to $c(H)$) bound on the diameter of a bridge-less asymmetric swap-equilibrium H in the weighted version of the game (with appropriately chosen weight function $c : V(H) \longrightarrow \mathbb{N}$) – in the following we use this approach.

We define and use the following notation. For every vertex $v \in V(H)$ let $T(v)$ denote the set of vertices $u \in V(G)$ for which a shortest u-H-path ends in v. Note that $v \in T(v)$, $T(v)$ induces a tree in G, and V is a disjoint union of $T(v)$, $v \in V(H)$. We define the weight function $c : V(H) \longrightarrow \mathbb{N}$ for vertices in H by setting $c(v) := |T(v)|$, and introduce the notation $c(H') := \sum_{v \in H'} c(v)$ for any $H' \subset H$. We note that $c(H) = n$. From now on we only consider H as a stand-alone, vertex-weighted, bridge-less graph. For $k \in \mathbb{N}$ and $u \in V(H)$ we define $B_k(u) := \{v \in V(H) : d_H(u, v) \leq k\}$ to be the *ball* of radius k and center u in H, and we define $S_k(u) := \{v \in V(H) : d_H(u, v) = k\}$ to be the *sphere* of radius k and center u in H. We further define $C_k := \min_{u \in V(H)} c(B_k(u))$ to be the minimum weighted size of any ball of radius k in H. For a vertex $u \in V(H)$ we denote its *eccentricity* in H by $D(u)$ (and recall that $D(u) = \max_{v \in V(H)} d(u, v)$).

We will need the following lemma, which shows that in asymmetric swap-equilibria a large (linear in n) number of vertices is far away from any given vertex u (the proof can be found in [8]).

Lemma 1. *For every vertex $u \in V(H)$ and $k < \frac{D(u)-1}{2}$ we have $c(B_k(u)) < \frac{D(u)+1}{2(D(u)-k)-1} \cdot n$.*

Corollary 3. *If $r := rad(H) > 14$ then for every vertex $u \in V(H)$ we have $c(B_{r/4}(u)) < \frac{3}{4}n$.*

In the following discussion, we bound the diameter of H using the “region-growing” technique of Demaine et al. [3] for the original network creation game (which showed an upper bound $2^{O(\sqrt{\log n})}$ on the price of anarchy). The details, however, differ significantly from the proofs in [3] due to the different definition of the games (players are only allowed to swap and not to buy new edges) and our new structural insights from Section 2.

Lemma 2. *If H has minimum degree $d(H) \geq n^{\varepsilon}$ for $\frac{4\lg 3}{\lg n} < \varepsilon < 1$, then for every vertex $u \in V(H)$ there is an edge $\{x, y\}$ induced by $B_{8/\varepsilon}(u)$ owned by x whose deletion increases x's usage cost by at most $\frac{20}{\varepsilon} n^{1-\varepsilon/2}$.*

Proof. We first show that there exists a vertex $v \in B_{2/\varepsilon}(u)$ which owns at least $\frac{n^{\varepsilon/2}}{2}$ edges in H. The main argument goes along the line "if there are m' edges among n' vertices, then there is a vertex that owns at least m'/n' edges (pigeon-hole principle)". First we consider the case when $|B_{k+1}(u)| \geq n^{\varepsilon/2}|B_k(u)|$ for every $k < 2/\varepsilon$. Clearly in this case, $|B_k(u)|$ is at least $n^{k\varepsilon/2}$ and therefore $|B_{2/\varepsilon}(u)| = |H|$. As H contains at least $\frac{|H|n^{\varepsilon}}{2}$ edges, there is a vertex $v \in B_{2/\varepsilon}(u) = H$ which owns at least $\frac{|H|n^{\varepsilon}}{2|H|} = n^{\varepsilon}/2$ edges.
Assume now that $|B_{k+1}(u)| < n^{\varepsilon/2}|B_k(u)|$ for some $k < 2/\varepsilon$. The ball $B_{k+1}(u)$ contains at least $\frac{n^{\varepsilon}}{2}|B_k(u)|$ edges (as all edges adjacent to vertices in $B_k(u)$ need to lie within $B_{k+1}(u)$). Therefore, there is a vertex in $B_{k+1}(u)$ which owns at least $\frac{n^{\varepsilon}}{2}|B_k(u)|/|B_{k+1}(u)| > \frac{n^{\varepsilon}}{2}|B_k(u)|/(n^{\varepsilon/2}|B_k(u)|) = \frac{n^{\varepsilon/2}}{2}$ edges.

We now investigate whether among the at least $\frac{n^{\varepsilon/2}}{2}$ edges which vertex $v \in B_{2/\varepsilon}(u)$ owns there is an edge whose deletion increases the usage cost by the claimed amount. For every edge $\{v, w\}$ which v owns, let $A(w)$ be the vertices x such that *every* shortest path from x to v goes via w. Observe that, as v owns at least $\frac{n^{\varepsilon/2}}{2}$ edges, there is an edge $\{v, w\}$ such that $c(A(w)) \leq n/(\frac{n^{\varepsilon/2}}{2}) = 2n^{1-\varepsilon/2}$. If this edge $\{v, w\}$ lies in a "short" cycle of length l, then deleting this edge would increase the usage cost of v by at most $l \cdot c(A(w))$. Consider vertices of $A(w)$ ordered in layers according to the distance to v. Let $k \in \mathbb{N}$ be the smallest index for which an edge between $S_k(v) \cap A(w)$ and $V(H) \setminus A(w)$ exists (such an edge indeed exists for some k as otherwise $\{v, w\}$ would be a bridge). If k is "small", then the edge lies in a "short" cycle of length $2k$ and we can bound the increase of the usage cost as suggested. Define for every $j \leq k$ the set $B_j := B_j(v) \cap A(w)$. In the case that $|B_{j+1}| \geq n^{\varepsilon/4}|B_j|$ for all $j < k$, we have $|B_k| \geq n^{(k\varepsilon)/4}$. But as $|B_k|$ is clearly upper bounded by n, we get $k < 4/\varepsilon$. Thus, k is a constant and deleting the edge $\{v, w\}$ increases the usage cost of v by at most $2k \cdot c(A(w)) < \frac{16n^{1-\varepsilon/2}}{\varepsilon}$. In the other case, let $j < k$ such that $|B_{j+1}| < n^{\varepsilon/4}|B_j|$. Note that, as $|B_j| < n$, $j < 4/\varepsilon$. In this case we do not consider deleting $\{v, w\}$ but instead we find another edge within B_j that can be deleted and which does not increase the usage cost of the owner of the edge too much. There are at least $\frac{n^{\varepsilon}}{2}|B_j|$ edges that are incident to vertices of B_j. If we subtract from these edges the edges that form a breadth-first search tree of B_{j+1} (there are at most $|B_{j+1}| \leq n^{\varepsilon/4}|B_j|$ of these), we obtain at least $\frac{n^{\varepsilon}-2n^{\varepsilon/4}}{2}|B_j|$ edges that are part of a "short" cycle of length no more than $(2j+2) \leq \frac{8}{\varepsilon} + 2$. Observe that $n^{\varepsilon} - 2n^{\varepsilon/4} = n^{3\varepsilon/4}(n^{\varepsilon/4} - 2n^{-2\varepsilon/4}) \geq n^{3\varepsilon/4}(n^{\varepsilon/4} - 2) > n^{3\varepsilon/4}(n^{\lg 3/\lg n} - 2) \geq n^{3\varepsilon/4}$. Therefore, there are at least $\frac{n^{3\varepsilon/4}}{2}|B_j|$ edges within B_{j+1} that are part of a "short cycle". There has to be a vertex $w \in B_{j+1}$ which owns at least $\frac{n^{3\varepsilon/4}/2|B_j|}{|B_{j+1}|} \geq \frac{n^{\varepsilon/2}}{2}$ of these edges. By the pigeon-hole principle, among these edges of w, there has to be one edge whose deletion increases the usage cost of w by at most $2(j+1)\frac{2n}{n^{\varepsilon/2}} < (\frac{16}{\varepsilon}+4)n^{1-\varepsilon/2} < \frac{20}{\varepsilon}n^{1-\varepsilon/2}$. □

Theorem 4. *If H has minimum degree $d(H) \geq n^{\varepsilon}$ for $\frac{4\lg 3}{\lg n} < \varepsilon < 1$, then there is a constant $C(\varepsilon) > 0$ (depending on ε) such that $diam(H) \leq C(\varepsilon)$.*

Proof. We will show that for every $k \leq \frac{r}{8} - 1$, where $r := \mathrm{rad}(H)$: $C_{3k+3+8/\varepsilon} > \frac{\varepsilon \cdot n^{\varepsilon/2}}{40} C_k$. Assuming this, the result follows immediately: Let $u \in V(H)$ be a vertex with eccentricity $D(u) = \mathrm{diam}(H)$ and let $\tilde{C} > 0$ be a constant such that $\tilde{C}k \geq 3k + 3 + 8/\varepsilon$. We have $c(B_{\tilde{C}k}(u)) \geq C_{3k+3+8/\varepsilon} > \frac{\varepsilon \cdot n^{\varepsilon/2}}{40} C_k$ for $k \leq \frac{r}{8} - 1$. Now, in the trivial case when n is at most the constant threshold $\left(\frac{40}{\varepsilon}\right)^{4/\varepsilon}$, then of course the diameter of H is at most this value (a constant). For the general case when n is larger than the threshold, we must have $\frac{r}{8} - 1 < \tilde{C}^{4/\varepsilon}$ as otherwise $c(B_{\frac{r}{8}-1}(u)) \geq c(B_{\tilde{C}^{4/\varepsilon}}(u)) \geq \left(\frac{\varepsilon n^{\varepsilon/2}}{40}\right)^{4/\varepsilon} C_1 \geq n$. Thus, in this case, the diameter of H is at most $2r < 16\tilde{C}^{4/\varepsilon} + 16$, a constant.

We now prove that $C_{3k+3+8/\varepsilon} > \frac{\varepsilon \cdot n^{\varepsilon/2}}{40} C_k$ for every $k \leq \frac{r-1}{4}$. Consider an arbitrary vertex $u \in V(H)$. By Lemma 2 there is an edge $\{x, y\}$ within $E(B_{8/\varepsilon}(u))$ owned by x whose deletion increases x's usage cost by at most $\frac{20n^{1-\varepsilon/2}}{\varepsilon}$. We select a maximal subset $\{x_1, \ldots, x_l\} \subset S_{2k+3}(x)$ subject to the condition $d(x_i, x_j) \geq 2k+1$ for every $i \neq j$. We assign every vertex in $S_{2k+3}(x)$ to the closest x_i, breaking ties arbitrarily. Let $S_{2k+3}(x) = \bigcup_{i=1}^{l} A_i$ be the obtained partition. We now prove that $l \geq \frac{\varepsilon \cdot n^{\varepsilon/2}}{40}$. We extend the partition and also assign each vertex $z \in V(H) \setminus B_{2k+2}(x)$ to one of the x_i, as follows: Pick any shortest path from z to x, and assign z to the same x_i as the (unique) vertex $w \in S_{2k+3}(x)$ which is contained in the path. After this step, $V(H) \setminus B_{2k+2}(x) = \bigcup_{i=1}^{l} A_i$ is the resulting partition. Consider vertex x and the swap of the edge $\{x, y\}$ with $\{x, x_i\}$ for arbitrary i: the distance of x to x_i decreases by $2k+2$ and hence, by the construction of A_i, the distance of x to the vertices of A_i decreases by at least 2. On the other hand, by Lemma 2, the swap increases x's usage cost by at most $\frac{20n^{1-\varepsilon/2}}{\varepsilon}$. Hence, as x cannot improve its usage cost by the swap (we are considering an asymmetric swap-equilibrium), $\frac{20n^{1-\varepsilon/2}}{\varepsilon} \geq 2c(A_i)$. As i was arbitrary, we have $l \cdot \frac{20n^{1-\varepsilon/2}}{\varepsilon} \geq 2\sum_{i=1}^{l} c(A_i)$. On the other hand, as $2k+2 \leq r/4$, we have by Corollary 3: $c(B_{2k+2}(x)) < 3n/4$, so $\sum_{i=1}^{l} c(A_i) = c(V(H) \setminus B_{2k+2}(x)) \geq n/4$. Therefore $l \cdot \frac{20n^{1-\varepsilon/2}}{\varepsilon} \geq 2\sum_{i=1}^{l} c(A_i) \geq n/2$ and hence $l \geq \frac{\varepsilon \cdot n^{\varepsilon/2}}{40}$.

By definition, $B_k(x_i) \cap B_k(x_j) = \emptyset$ for every $i \neq j$. Hence $c(\bigcup_{i=1}^{l} B_k(x_i)) = \sum_{i=1}^{l} c(B_k(x_i)) \geq l \cdot C_k$. Furthermore, for every $1 \leq i \leq l$, we have $d(u, x_i) \leq d(u, x) + d(x, x_i) \leq 2k + 3 + 8/\varepsilon$, so vertex u has a path of length at most $3k + 3 + 8/\varepsilon$ to every vertex in $B_k(x_i)$. Therefore $c(B_{3k+3+8/\varepsilon}(u)) \geq \sum_{i=1}^{l} c(B_k(x_i)) > l \cdot C_k \geq \frac{\varepsilon n^{\varepsilon/2}}{40} C_k$. Hence, as $u \in V(H)$ was chosen arbitrarily, $C_{3k+3+8/\varepsilon} > \frac{\varepsilon n^{\varepsilon/2}}{40} C_k$. □

Together with Corollary 1 resp. Corollary 2 this implies a logarithmic bound on the diameter of asymmetric swap-equilibria, respectively a constant bound on the diameter of swap equilibria:

Corollary 4. *For any ϵ and any non-tree asymmetric swap equilibrium G, the following holds. If the unique 2-edge-connected component H of G has minimum degree $d(H) \geq n^{\varepsilon}$ and $\frac{4 \lg 3}{\lg n} < \varepsilon < 1$ then diam$(G) = \mathcal{O}(\lg n)$.*

Corollary 5. *For every constant $0 < \varepsilon < 1$ there is a constant $C(\varepsilon)$ such that the following holds. The diameter of any graph G, where $d(H) \geq n^{\varepsilon}$ and $\frac{4 \lg 3}{\lg n} < \varepsilon$ (and H is the unique 2-edge-connected component of G), is at most $C(\varepsilon)$.*

Inspired by Theorem 4 and by the fact that one could not find any bridge-less equilibrium graph of diameter greater than 4, we conjecture:

Conjecture 1. There exists a constant $C > 0$ such that the diameter of the unique 2-edge connected component of any asymmetric swap-equilibrium is smaller than C. In particular, every asymmetric swap-equilibrium has diameter $\mathcal{O}(\log n)$ and every swap equilibrium has diameter $\leq C + 4$.

References

1. Albers, S., Eilts, S., Even-Dar, E., Mansour, Y., Roditty, L.: On Nash equilibria for a network creation game. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, pp. 89–98 (2006)
2. Alon, N., Demaine, E.D., Hajiaghayi, M., Leighton, T.: Basic network creation games. In: Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA, pp. 106–113 (2010)
3. Demaine, E.D., Hajiaghayi, M., Mahini, H., Zadimoghaddam, M.: The price of anarchy in network creation games. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC, pp. 292–298 (2007)
4. Ehsani, S., Fazli, M., Mehrabian, A., Sadeghian Sadeghabad, S., Safari, M., Saghafian, M., ShokatFadaee, S.: On a bounded budget network creation game. In: Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA, pp. 207–214 (2011)
5. Fabrikant, A., Luthra, A., Maneva, E., Papadimitriou, C.H., Shenker, S.: On a network creation game. In: Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing, PODC, pp. 347–351 (2003)
6. Lenzner, P.: On Dynamics in Basic Network Creation Games. In: Persiano, G. (ed.) SAGT 2011. LNCS, vol. 6982, pp. 254–265. Springer, Heidelberg (2011)
7. Mihalák, M., Schlegel, J.C.: The Price of Anarchy in Network Creation Games Is (Mostly) Constant. In: Kontogiannis, S., Koutsoupias, E., Spirakis, P.G. (eds.) SAGT 2010. LNCS, vol. 6386, pp. 276–287. Springer, Heidelberg (2010)
8. Mihalák, M., Schlegel, J.C.: Asymmetric swap-equilibrium: A unifying equilibrium concept for network creation games. Tech. Rep. 764, Department of Computer Science, ETH Zurich (June 2012), https://www1.ethz.ch/inf/research/disstechreps/techreports

Between Tree Patterns and Conjunctive Queries: Is There Tractability beyond Acyclicity?

Filip Murlak, Michał Ogiński, and Marcin Przybyłko

Institute of Informatics, University of Warsaw
fmurlak@mimuw.edu.pl, {M.Oginski,M.Przybylko}@students.mimuw.edu.pl

Abstract. In static analysis of queries over trees in the presence of schemas, there is an exponential complexity gap between conjunctive queries (CQs, positive existential first-order formulae without disjunction) and tree patterns (tree-like acyclic CQs). Motivated by applications in XML data management, we consider various restrictions of CQs that bring their complexity down to that of tree patterns. Most importantly, we show that vertical tree patterns can be costlessly extended with full horizontal CQs over children. We also consider restricted classes of schemas and show that under disjunction-free schemas the complexity of static analysis sometimes drops dramatically.

1 Introduction

Static analysis is a common name used in database theory for problems that do not deal with data, but only with queries. Such problems are often part of complex data management tasks, like data integration and data exchange [15, 19]. Most important static analysis problems include satisfiabiliy (Given a query q, is there a database D such that the returned set of tuples $q(D)$ is nonempty?) and query containment (Given q_1, q_2, does $q_1(D) \subseteq q_2(D)$ hold for each D?). As for first order logic these problems are undecidable, restricted query languages are considered. For relational databases, conjunctive queries (CQs, positive existential formulae without disjunction) and their unions (UCQs) are used most widely. The reason is a relatively low cost of static analysis [12], and expressive power meeting most typical needs (select-from-where SQL queries). In contrast, in XML static analysis, where problems are relativized to XML trees accepted by a given schema (often modelled as a tree automaton), the complexity of full CQs, using child and descendant relations, and sibling order, is prohibitively high [8, 9, 16]. As a remedy, more restrictive languages of acyclic CQs and tree patterns (tree-like acyclic CQs) were introduced. For instance, literature on XML data exchange and metadata management considers almost exclusively tree patterns [1–3, 14]. A fine complexity analysis for CQs and UCQs over XML trees would be useful in designing richer formalisms, based on intermediate classes of queries.

Most research on static analysis for queries over XML trees was done for fragments of XPath 1.0, which is a language allowing only acyclic queries [5, 18, 20, 21, 24, 25], or XPath 2.0, which allows path intersection, but not

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 705–717, 2012.

arbitrary joins [11, 17]. As has been observed by Gottlob, Koch, and Shulz, each CQ on trees can be translated to a union of exponentially many polynomial tree patterns [16]. This gives an upper bound on the complexity of containment exponentially higher then that for tree patterns: 2ExpTime in general and ExpSpace under non-recursive schemas (where the depth of trees is bounded by the size of the schema), while for tree patterns they are ExpTime and PSpace, respectively [5, 20, 24]. Björklund, Martens, and Schwentick show that the exponential gap cannot be avoided in general, as even containment of CQs using only child and descendant relation is 2ExpTime-complete, and ask if there are more manageable classes of CQs, other then acyclic CQs [9]. We are most interested in results of the form: under certain restrictions, the complexity of containment of UCQs is the same as that of unions of tree patterns. For example, if only child relation is available, the general case reduces to the acyclic case, as each CQ can be rewritten as a single tree pattern of linear size (cf. [4]).

We focus on the restrictions most commonly studied in XML data exchange and metadata management: non-recursive or disjunction-free schemas (cf. nested-relational DTDs [2, 3, 6]), and limited use of horizontal or vertical relations. We first prove that, over words, containment of UCQs is PSpace-complete, just like for unions of tree patterns (Sect. 4). Then we apply these results to trees (Sect. 5) and show that the complexities match for a fairly general class of "forest-like" UCQs, combining vertical tree patterns with arbitrary horizontal CQs over children. This is further exploited to prove the same for

- UCQs that do not use the descendant relation;
- UCQs specifying labels of all mentioned nodes, under non-recursive DTDs.

Finally in Sect. 6 we show that under disjunction-free schemas the containment of UCQs without the next-sibling relation is in coNExpTime and PSpace-hard, and if additionally schemas are non-recursive, it is on the second level of the polynomial hierarchy; with next-sibling, the complexity does not drop.

We work exclusively with Boolean queries; as explained in [9], this is not a restriction. Due to space limitations some arguments are omitted. For more details see the appendix available at `www.mimuw.edu.pl/~fmurlak/papers/patsat.pdf`.

2 Preliminaries

XML documents and trees. We model XML documents as unranked labelled trees. Formally, a *tree* over a finite labelling alphabet Γ is a relational structure $\mathcal{T} = \langle T, \downarrow, \downarrow^+, \rightarrow, \xrightarrow{+}, (a^{\mathcal{T}})_{a\in\Gamma}\rangle$, where

- the set T is an unranked tree domain, i.e., a prefix-closed subset of $\mathbb{N}^*$ such that $n \cdot i \in T$ implies $n \cdot j \in T$ for all $j < i$;
- the binary relations $\downarrow$ and $\rightarrow$ are the child relation ($n \downarrow n \cdot i$) and the next-sibling relation ($n \cdot i \rightarrow n \cdot (i + 1)$);
- $\downarrow^+$ and $\xrightarrow{+}$ are transitive closures of $\downarrow$ and $\rightarrow$;
- $(a^{\mathcal{T}})_{a\in\Gamma}$ is a partition of the domain T into possibly empty sets.

We write $|\mathcal{T}|$ to denote the number of nodes of tree $\mathcal{T}$. The partition $(a^{\mathcal{T}})_{a\in\Gamma}$ defines a labelling of the nodes of $\mathcal{T}$ with elements of Γ, denoted by $\ell_{\mathcal{T}}$.

Automata and DTDs. The principal schema language we use are tree automata, abstracting Relax NG [22, 23]. We are using a variant in which the state in a node v depends on the states in the previous sibling and the last child of v. Such automata are equivalent to standard automata on unranked trees, as explained in [23]. Formally, an automaton is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$, where Σ is the labelling alphabet (the set of element types in our case), Q is the state space with the initial state q_0 and final states F, and $\delta \subseteq Q \times Q \times \Sigma \times Q$ is the transition relation. A *run* of $\mathcal{A}$ over a tree $\mathcal{T}$ is a labelling ρ of the nodes of $\mathcal{T}$ with the states of $\mathcal{A}$ such that for each node v with children $v \cdot 0, v \cdot 1, \ldots, v \cdot k$ and previous sibling w, $\big(\rho(w), \rho(v \cdot k), \ell_{\mathcal{T}}(v), \rho(v)\big) \in \delta$. If v has no previous sibling, $\rho(w)$ in the condition above is replaced with q_0. Similarly, if v has no children, $\rho(v \cdot k)$ is replaced with q_0. The language of trees *recognized* by $\mathcal{A}$, denoted by $L(\mathcal{A})$, consists of all trees admitting an *accepting* run of $\mathcal{A}$, i.e. a run that assigns one of the final states to the root.

A simpler schema language is provided by DTDs. A *document type definition* (DTD) over a labelling alphabet Γ is a pair $D = \langle r, P_D \rangle$, where $r \in \Gamma$ is a distinguished root symbol and P_D is a function assigning regular expressions over $\Gamma - \{r\}$ to the elements of Γ, usually written as $\sigma \to e$, if $P_D(\sigma) = e$. A tree $\mathcal{T}$ *conforms to* a DTD D, denoted $\mathcal{T} \models D$, if its root is labelled with r and for each node s in $\mathcal{T}$ the sequence of labels of its children is in the language of $P_D(\ell_{\mathcal{T}}(s))$. The set of trees conforming to D is denoted by $L(D)$. It is well known (and easy to see) that there is a PTime translation from DTDs to automata.

We shall often consider *non-recursive* schemas, DTDs or automata. A DTD D is non-recursive if in every tree conforming to D each path contains each label at most once. A schema given by a tree automaton is non-recursive if in each run (accepting or not) each path contains each state at most once. The height of trees conforming to non-recursive schemas is bounded by the size of the schema.

CQs and Patterns. A conjunctive query (CQ) over alphabet Γ is a formula of first order logic using only conjunction and existential quantification, over unary predicates $a(x)$ for $a \in \Gamma$ and binary predicates $\downarrow, \downarrow^+, \to, \xrightarrow{+}$ (referred to as *child*, *descendant*, *next sibling*, and *following sibling*, respectively). Since we work only with Boolean queries, to avoid unnecessary clutter we often skip the quantifiers, assuming that all variables are by default quantified existentially.

An alternative way of looking at CQs is via patterns. A *pattern* π over Γ can be presented as $\pi = \langle V, E_c, E_d, E_n, E_f, \ell_\pi \rangle$ where ℓ_π is a partial function from V to Γ, and $\langle V, E_c \cup E_d \cup E_n \cup E_f \rangle$ is a finite graph whose edges are split into child edges E_c, descendant edges E_d, next-sibling edges E_n, and following-sibling edges E_f. By $|\pi|$ we mean the size of the underlying graph.

We say that a tree $\mathcal{T} = \langle T, \downarrow, \downarrow^+, \to, \xrightarrow{+}, (a^{\mathcal{T}})_{a \in \Gamma} \rangle$ *satisfies* a pattern $\pi = \langle V, E_c, E_d, E_n, E_f, \ell_\pi \rangle$, denoted $\mathcal{T} \models \pi$, if there exists a homomorphism $h \colon \pi \to \mathcal{T}$, i.e., a function $h : V \to T$ such that

- $h : \langle V, E_c, E_d, E_n, E_f \rangle \to \langle T, \downarrow, \downarrow^+, \to, \xrightarrow{+} \rangle$ is a homomorphism of relational structures; and
- $\ell_{\mathcal{T}}(h(v)) = \ell_\pi(v)$ for all v in the domain of ℓ_π.

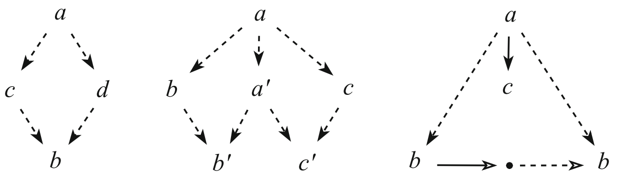


Fig. 1. Typical patterns. Void and solid heads indicate horizontal and vertical order, respectively. Dashed lines indicate transitive closure. The leftmost pattern can be expressed using path intersection operator from XPath 2.0, but the middle one cannot.

Each pattern can be seen as a CQ, and vice versa. In what follows we use the terms "pattern" and "CQ" interchangeably. *Tree patterns* are patterns whose underlying graph is a directed tree with edges pointing from parents to children.

Containment and Satisfiability. We focus on the following satisfiability problem.

PROBLEM: BC-SAT
INPUT: Boolean combination of patterns φ, schema $\mathcal{S}$ (automaton or DTD).
QUESTION: Is there a tree $\mathcal{T} \in L(\mathcal{S})$ such that $\mathcal{T} \models \varphi$?

For $\sigma \subseteq \{\downarrow, \downarrow^+, \rightarrow, \overset{+}{\rightarrow}, \_\}$ we write BC-SAT(σ) to denote BC-SAT restricted to patterns which use only the axes listed in σ. If the wildcard symbol, $\_$, is not present in σ, the labelling functions in patterns are required to be total, or in other words that each variable must occur in an atom of the form $a(x)$ for some $a \in \Gamma$. We use the following abbreviations: $\Downarrow$ for $\downarrow, \downarrow^+, \_$; and $\Rightarrow$ for $\rightarrow, \overset{+}{\rightarrow}, \_$.

Containment for UCQs is inter-reducible with non-satisfiability for Boolean combinations of the form $\pi \wedge \neg\pi_1 \wedge \neg\pi_2 \wedge \cdots \wedge \neg\pi_k$. Validity of a query is equivalent to non-satisfiability of its negation. Most of our lower bounds only use conjunctions of negations of patterns, thus giving dual lower bounds for validity (and containment).

3 Basic Complexity Bounds

In this section we briefly summarize known complexity bounds and establish some new bounds in the case of non-recursive schemas. The complexity of BC-SAT for tree patterns follows immediately from the results on XPath satisfiability and containment.

Theorem 1 ([5, 11, 20, 24]). *For tree patterns,* BC-SAT($\Downarrow, \Rightarrow$) *is* EXPTIME-*complete and* PSPACE-*complete under non-recursive schemas. The lower bounds hold even for containment of unions of tree patterns using only* $\downarrow$*.*

The EXPTIME upper bound follows from the translation of downward XPath to automata [24], later extended to cover horizontal axes in [11] (also implicit in [20]). The PSpace upper bound follows from [5]. The lower bounds in [5, 24] rely on the availability of wildcard (or disjunction inside XPath expressions), but they can be strengthened to queries using only ↓.

As we have mentioned, for CQs the bounds are exponentially worse.

Theorem 2 ([9]). *For arbitrary patterns,* $\mathsf{BC\text{-}SAT}(\Downarrow, \Rightarrow)$ *is* 2ExpTime*-complete. The lower bound holds already for validity of a single CQ using only* $\Downarrow$*.*

To complete the background for the main results of this paper, described in Sections 4–6, we now settle the complexity of $\mathsf{BC\text{-}SAT}$ under non-recursive schemas. The complexity drops only slightly, but the cost of → begins to show.

Theorem 3. *Under non-recursive schemas* $\mathsf{BC\text{-}SAT}(\Downarrow)$ *and* $\mathsf{BC\text{-}SAT}(\Downarrow, \overset{+}{\rightarrow})$ *is* NExpTime*-complete and* $\mathsf{BC\text{-}SAT}(\Downarrow, \Rightarrow)$ *is* ExpSpace*-complete. The lower bounds hold already for conjunctions of negated patterns.*

The ExpSpace upper bound follows immediately by translation to tree patterns. The NExpTime upper bound is obtained via a linearly-branching model property, which relies on the fact that an unsatisfied pattern without → never becomes satisfied when a subtree is deleted. The lower bounds use an ingenious pattern construction from [9].

4 CQs over Words

In the classification sketched out in the previous section, horizontal CQs were to some extent drowned in the overall complexity of patterns. We shall have a closer look at them now: we restrict our models to words. We show that under this restriction the complexity of CQs matches that of tree patterns (Theorem 4); in the next section we show how this can be applied to the tree case.

Our main building block is a procedure $\textsc{Match}_\pi$ associated with each pattern π. The procedure takes as input a word w and checks if $w \models \pi$. It reads w letter by letter, possibly storing some information in the working memory, polynomial in $|\pi|$ and independent of w (it can be seen as a DFA, exponential in π). The procedure looks for the *earliest (leftmost) matching* of π in w, as defined below. We note that earliest matchings were previously used in [7] for tree patterns.

We use $\leq$ and $+1$ for the standard order and successor on the positions of words (an initial segment of natural numbers), and define homomorphisms just like for trees. Whenever we write $h : \pi \to w$, we implicitly assume that h is a homomorphism.

Definition 1. *Let* $g, h \colon \pi \to w$ *be two homomorphisms.*

- *We write* $g \leq h$ *if* $g(v) \leq h(v)$ *for each vertex* v *of* π*.*
- *We define* $\min(g,h) \colon \pi \to w$ *as* $\min(g,h)(v) = \min(g(v), h(v))$*.*

Lemma 1. *Let* w *be a word satisfying a* $\Rightarrow$*-pattern* π*.*

1. *For all* $g, h \colon \pi \to w$, $\min(g, h)$ *is a homomorphism.*
2. *There exists* $h_{\min} \colon \pi \to w$ *such that* $h_{\min} \leq h$ *for all* $h \colon \pi \to w$.
3. *For each set* X *of vertices of* π *and each* $h \colon \pi \to w$ *there is a* $\hat{h} \colon \pi \to w$ *extending* $h|_X$ *such that* $\hat{h} \leq h'$ *for each* $h' \colon \pi \to w$ *extending* $h|_X$.

We call the unique $h_{\min}$ from Lemma 1 the *earliest matching* of π in w.

$\textsc{Match}_\pi(w)$ works with components of π, called *firm subpatterns*, described in Definition 3.

Definition 2. *A* →-component *of* π *is a maximal connected subgraph of* →*-graph of* π*. In the* graph of →-components of π*, denoted* G_π*, there is an edge from a* →*-component* π_1 *to a* →*-component* π_2 *if there is a* $\xrightarrow{+}$ *edge in* π *from a vertex of* π_1 *to a vertex of* π_2*.*

Definition 3. *A pattern* π *is* firm *if* G_π *is strongly connected. In general, each strongly connected component* X *of* G_π *defines a* firm subpattern *of* π*: the subgraph of* π *induced by the vertices of* →*-components contained in* X*. The* DAG of firm subpatterns *of* π*, denoted* F_π*, is the standard DAG of strongly connected components of* G_π*.*

For example, the pattern in Fig. 2 on page 712 is firm, but has three →-components.

The matching procedure $\textsc{Match}_\pi(w)$ works as follows:

- it reads the input word w from left to right trying to match firm subpatterns of π in the topological order given by F_π;
- for each firm subpattern it finds the earliest matching that does not violate the $\xrightarrow{+}$ edges connecting it with previously matched firm subpatterns.

Since we are proceeding in the topological order, each firm subpattern is processed after all its predecessors have been matched. Hence, the algorithm always finds a correct homomorphism or none at all. Completeness of the algorithm follows from the lemma below by straightforward induction (where Y is the union of previously matched firm patterns and X is obtained by adding a new one).

Lemma 2. *Let* $h \colon \pi \to w$ *be the earliest matching and let* X *be a set of vertices of* π *such that no edge enters* X *from the outside, and the only edges leaving* X *are* $\xrightarrow{+}$*. For each* $Y \subseteq X$*, if* $g \colon \pi|_X \to w$ *is the least homomorphism extending* $h|_Y$ *to* X*, then* $h|_X = g$*.*

Now we need to bound the memory used by $\textsc{Match}_\pi(w)$. We claim that the algorithm only needs to remember last $|\pi|$ symbols read (plus the matching constructed so far, restricted to this suffix). It is straightforward to check that each homomorphic image of a firm pattern π_0 is a subword of length at most $|\pi_0|$. Based on this observation, we prove the claim. For $i \leq |w|$, let Π_i be the set of firm subpatterns of π matched by $\textsc{Match}_\pi(w)$ after processing the first i symbols of w. Note that if a position j is not touched by the matching, all firm subpatterns matched before this position are in Π_j. By pigeon-hole principle, there is a position j between $i - |\pi|$ and i that is not touched by the matching. By the previous comment, all patterns from $\Pi_i \setminus \Pi_{i-1}$ are matched between j and i. It follows that $\textsc{Match}_\pi(w)$ only needs to remember the last $|\pi|$ symbols.

Using the matching procedure we prove the main result of this section.

Theorem 4. *On words* $\mathsf{BC\text{-}SAT}(\Rightarrow)$ *is* PSpace*-complete, with hardness already for conjunctions of negated tree patterns using only* $\to, \_$ *or CQs using only* $\to, \overset{+}{\to}$.

Proof. To check if a Boolean combination φ is satisfiable in a word accepted by an automaton $\mathcal{A}$, we non-deterministically generate letters of a word $w \in L(\mathcal{A})$ and feed with them $\textsc{Match}_\pi$ for each π used in φ. We accept if the split into matched and unmatched patterns satisfies φ. To prevent looping, we count the number of letters and stop when we reach certain threshold, single exponential in $|\varphi|$. To establish the threshold, recall that $\textsc{Match}_\pi$ can be seen as a DFA, exponential in $|\pi|$. The product automaton corresponding to all running copies of the matching procedure is single exponential in φ. The threshold can be set to the size of the product automaton. By Savitch theorem we can eliminate non-determinism from this algorithm.

For the lower bound we give a reduction from the following tiling problem, which is known to be PSpace-complete: Given a set of tiles $T = \{t_1, t_2, \ldots, t_k\}$, relations $H, V \subseteq T \times T$, and a number n in unary, decide if there is a number m and an $m \times n$ matrix $(a_{i,j})$ with entries from T such that $a_{1,1} = t_1$, $a_{m,n} = t_k$, $(a_{i,j}, a_{i,j+1}) \in H$ for $1 \leq i \leq m$, $1 \leq i < n$ and $(a_{i,j}, a_{i+1,j}) \in V$ for $1 \leq i < m$, $1 \leq i \leq n$. In fact we give the reduction from the following linearised tiling to which the original problem can be easily reduced: Given T, H, V, n, decide if there is a sequence of tiles $s_1 s_2 \ldots s_\ell$ such that $s_1 = t_1$, $s_\ell = t_k$, $(s_i, s_{i+1}) \in H$ for all $i \leq \ell - 1$, and $(s_i, s_{i+n}) \in V$ for all $i \leq \ell - n$.

Let an instance of the linearised tiling problem be T, H, V, n. If wildcard is available, we can assume our alphabet is $T \cup \{r\}$ and take the DTD $r \to t_0 T^* t_k$ and the following combination of patterns:

$$\bigwedge_{(t_i,t_j)\notin H} \neg\exists x\, \exists y\ (x \to y) \wedge t_i(x) \wedge t_j(y) \wedge \bigwedge_{(t_i,t_j)\notin V} \neg\exists x\, \exists y\ (x \to^n y) \wedge t_i(x) \wedge t_j(y)\,.$$

Without wildcard we cannot express $\to^n$, but we can circumvent this obstacle using $\overset{+}{\to}$ if we modify our encoding properly. We encode the tile t_i as the word

$$w_i = \triangleright \bar{a} a^i b a^j \bar{a} \triangleleft$$

with $\bar{a} = aa^k$ and $i + j + 1 = k$. For the DTD we take $r \to w_0 W^* w_k$, where $W = \{w_i \mid i = 1, 2, \ldots, k\}$. The patterns are replaced with

$$x_1 \xrightarrow{w_i} x_1' \to x_2 \xrightarrow{w_j} x_2'\,,$$

$$x_1 \xrightarrow{w_i} x_1' \to x_2 \xrightarrow{w_*} x_2' \to \ldots \to x_n \xrightarrow{w_*} x_n' \to x_{n+1} \xrightarrow{w_j} x_{n+1}'\,,$$

where $x \xrightarrow{w_i} x'$ is a pattern that says that the segment of the word from position x to x' is w_i and $x \xrightarrow{w_*} y$ is the following pattern (see also Fig. 2)

$$(x \xrightarrow{\triangleright\bar{a}} x'') \wedge (x' \xrightarrow{\bar{a}b\bar{a}} y') \wedge (y'' \xrightarrow{\bar{a}\triangleleft} y) \wedge (x \overset{+}{\to} x' \overset{+}{\to} x'') \wedge (y'' \overset{+}{\to} y' \overset{+}{\to} y)\,. \qquad \square$$

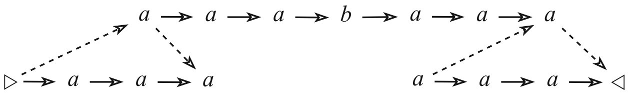


Fig. 2. Pattern $x \xrightarrow{w_*} y$ for $k = 2$

Thus BC-SAT($\Rightarrow$) is PSpace-complete if either wildcard or $\xrightarrow{+}$ and arbitrary joins are allowed. If we forbid wildcards and restrict the use of joins, the complexity drops to NP. Using Match$_\pi$ and the following observation, we prove the upper bound for a relatively large class of patterns, extending tree patterns.

Lemma 3. *For all $w_1, w_2, \ldots, w_n \in \Gamma^*$ there is a linear-size deterministic automaton recognizing $\bigcup_{i=1}^{n} \Gamma^* w_i \Gamma^*$. One can compute it in* PTime.

Proof. States of the automaton are prefixes of w_i's. After reading u, the state is the longest prefix that is a suffix of u. If the prefix is the whole word w_i, the automaton moves to a distinguished accepting state. □

Theorem 5. *For patterns whose firm sub-patterns do not contain $\xrightarrow{+}$, the problem* BC-SAT($\rightarrow, \xrightarrow{+}$) *on words is* NP*-complete.*

Proof. The lower bound is proved in [9]. To get the upper bound, we prove a polynomial model property. Given that $\Rightarrow$-patterns can be evaluated in PTime, the proposition follows.

Let $w \in L(\mathcal{A})$ be a word satisfying a Boolean combination φ. For each pattern π in φ consider its *earliest partial matching*, i.e., the partial matching computed by Match$_\pi$. Clearly, π is satisfied if and only if its earliest partial matching is total. It suffices to show that the segments of w outside of the partial matches can be chosen small, without changing the matches.

Suppose that $w = u_1 v u_2$ and v is not touched by the partial matchings. A partial matching is earliest if and only if each firm sub-pattern π_0 is matched at its first occurrence after the *launching point*: the latest position i such that matching π_0 at i (regardless of labels) violates some $\xrightarrow{+}$ edge entering π_0. When shortening v we only need to make sure that we do not introduce an occurrence of a subpattern between its launching point and its original match in w. For subpatterns matched in u_1 changing v makes no difference. Suppose π_0 is matched in u_2. Where can the launching point of π_0 be? If it is enforced by a sub-pattern matched in u_1, it is in u_1. If it is enforced by a subpattern matched in u_2, it is either in u_2 or within the last $|\pi_0|$ positions of v.

Let $\pi_1, \pi_2, \ldots, \pi_k$ be all sub-patterns matched in u_2 whose launching points are in u_1. Since they contain no $\xrightarrow{+}$ nor _, they can be turned into single words by merging along the $\rightarrow$ edges. Let $\mathcal{B}$ be the deterministic automaton accepting words that contain some π_i (Lemma 3). Let $v = v_1 v' v_2$, where $|v_1| = |v_2|$ is equal to the maximal size of a firm subpattern. By standard pumping we can shorten v' to at most $\|\mathcal{A}\| \cdot \|\mathcal{B}\|$, without introducing new occurrences of π_i's in

vu_2. Since we are not touching v_1, we do not introduce new occurrences of π_i's in the whole word w. Similarly, since we are not touching v_2, the patterns whose launching points are in v_2u_2 are not influenced either. □

5 Back to Trees

We now lift the restriction on models and see what happens for trees. We have already seen that BC-SAT for full CQs is exponentially harder than for tree patterns: 2ExpTime versus ExpTime, and ExpSpace versus PSpace under non-recursive schemas (Theorems 1–3). Here we consider several restrictions on CQs and schemas that lower the complexity of CQs to that of tree patterns.

We show first that, for vertical tree patterns extended with arbitrary horizontal CQs over siblings, our PSpace algorithm for BC-SAT on words can be incorporated into the procedures for tree patterns without increasing their complexity. (Allowing joins with arbitrary horizontal CQs would immediately violate the intended tree structure of the vertical part of the pattern.) We say that a pattern is *forest-like* if its $\Downarrow$-subgraph is a disjoint union of trees and all vertical edges coming to the same connected $\Rightarrow$-subpattern originate in the same vertex.

Theorem 6. *For forest-like patterns,* BC-SAT$(\Downarrow, \Rightarrow)$ *is* ExpTime*-complete, and under non-recursive schemas it is* PSpace*-complete.*

Proof. For a forest-like pattern π we shall construct an equivalent deterministic automaton $\mathcal{A}_\pi$, whose states and transitions can be generated in PSpace. (Recall that a tree automaton is (bottom-up) deterministic, if for all $q_1, q_2 \in Q$ and $a \in \Sigma$ there exists exactly one state q such that $(q_1, q_2, a, q) \in \delta$.) Using this construction one can reduce BC-SAT$(\Downarrow, \Rightarrow)$ to nonemptiness of tree automata in PSpace. Both upper bounds follow, since nonemptiness of $\mathcal{A}$ over trees of depth d can be tested in space $\mathcal{O}(d \cdot \log \|\mathcal{A}\|)$, and over arbitrary trees in PTime.

A *horizontal component* of π is a connected component of the $\Rightarrow$-subgraph of π. Let $H_\pi = \langle V_\pi, \downarrow, \downarrow^+ \rangle$ be a graph over horizontal components of π, where edge $\pi_1 \downarrow \pi_2$ is present if $x \downarrow y$ for some $x \in \pi_1$ and $y \in \pi_2$, and $\pi_1 \downarrow^+ \pi_2$ is present if $x \downarrow^+ y$ for some $x \in \pi_1$ and $y \in \pi_2$, but there is no edge $\pi_1 \downarrow \pi_2$. Since π is forest-like, this graph is a forest. The subtree of H_π rooted at π_1 defines a subpattern of π, denoted by $(\pi_1)_\Downarrow$. We call such subpatterns *subtrees of* π.

The automaton $\mathcal{A}_\pi$, after reading the sequence of children of a node v, passes to v information about subtrees of π that were matched in the children of v and those that were matched in the children of some descendant of v. The automaton accepts, if the information passed from the root says that π was matched. To compute the information to be passed to v the automaton needs to aggregate the information passed from v's grandchildren to their parents. This is done by a modified version of Match working over an extended alphabet, described below.

A subtree $(\pi_1)_\Downarrow$ of π can be viewed as a horizontal pattern obtained from π_1 by including in the label of each vertex x the information about the subtrees of π to which x is connected by $\downarrow$ and $\downarrow^+$ edges. At each step Match is fed with a symbol that consists of the label of a tree node u and the information passed to

u from its sequence of children. (At the leaf level of T this information is void and Match works just like for words.) Match is only altered in this way that a vertex labelled with an extended label σ can be matched in a position labelled with an extended label τ if the original labels agree and all patterns listed in σ are also listed in τ (keeping the distinction between patterns connected by $\downarrow$ and $\downarrow^+$). It is straightforward to check that this does not influence correctness of Match. Observe that the extended alphabet is exponential, but each symbol can be stored in polynomial memory. Hence, Match still works in memory polynomial in the size of the pattern.

This procedure can be easily implemented by an exponential deterministic tree automaton. Within a sequence of children, $\mathcal{A}_\pi$ behaves like the automaton implementing Match($\tilde{\pi}$), where $\tilde{\pi}$ is the disjoint union of all subtrees of π. It reads the extended label from the label of the current child u and the state coming from the children of u. When the last child is read, the information about matched subtrees of π is complete and can be passed up, to the parent. □

From this result we obtain further upper bounds. For purely horizontal patterns, a standard pumping argument allows us to bound the height of the witnessing tree by the size of the schema, and with some care the algorithm for non-recursive schemas can be used even if the original schema is recursive.

Corollary 1. BC-SAT($\Rightarrow$) *is* PSpace-*complete.*

Furthermore, as observed in [16], each pattern using no $\downarrow^+$ can be turned in PTime into an equivalent forest-like pattern by simply merging each pair of vertices that have outgoing $\downarrow$ edges to the same connected $\Rightarrow$-subpattern. Hence, we immediately get the following corollary (hardness from Theorem 1).

Corollary 2. BC-SAT($\downarrow, \Rightarrow$) *is* ExpTime-*complete and* PSpace-*complete under non-recursive schemas.*

Finally, with a little more effort one can prove that in the presence of a non-recursive DTD the same holds for patterns that do not use wildcard.

Corollary 3. BC-SAT($\downarrow, \downarrow^+, \rightarrow, \xrightarrow{+}$) *is* PSpace-*complete under non-rec. DTDs.*

The lower bound follows from Theorem 1 and the upper bound relies on the fact that in the presence of a non-recursive DTD labels come in a fixed order in the paths. For non-recursive tree automata this is no longer the case. In fact, under such schemas one can carry over the lower bounds of Theorem 3 to the case without wildcard.

6 Disjunction-Free DTDs

Theorem 3 shows that under non-recursive schemas BC-SAT does not get much easier. We now introduce another restriction, often used in combination with non-recursivity in complex data management tasks [2, 3]: we limit the use of disjunction. A DTD is *disjunction-free* if its regular expressions use only concatenation, Kleene star and the operator $\alpha^{\leq m} = (\varepsilon \,|\, \alpha \,|\, \alpha^2 \,|\, \ldots \,|\, \alpha^m)$.

BC-SAT under disjunction-free DTDs is not easier unless → is forbidden. Indeed, using → we can simulate full DTDs, e.g., a production $a \to \alpha \,|\, \beta$ can be simulated by $a \to \sharp(\triangleright\alpha\triangleleft)^*(\triangleright\beta\triangleleft)^*\sharp$, with conjunct $\neg\exists x\, \exists y\, \big(\sharp(x) \to \sharp(y)\big) \wedge \neg\exists x\, \exists y\, \big(\triangleleft(x) \to \triangleright(y)\big)$ added to the combination tested for satisfiability.

If → is forbidden, the complexity under disjunction-free non-recursive DTDs drops to low levels of the polynomial hierarchy, compared to NExpTime for non-recursive DTDs allowing disjunction (Theorem 3).

Theorem 7. *Under non-recursive disjunction-free DTDs* $\mathsf{BC\text{-}SAT}(\Downarrow, \overset{+}{\to})$ *is* Σ_2P*-complete and* NP*-complete for tree patterns.*

The problem is Σ_2P-hard already for Boolean combinations of the form $\pi_1 \wedge \neg\pi_2$ where π_1 is a pattern with a single node, but without π_1 it is coNP-complete.

If the non-recursivity restriction is lifted the complexity is still (potentially) below the general 2ExpTime lower bound.

Theorem 8. *Under disjunction-free DTDs* $\mathsf{BC\text{-}SAT}(\Downarrow, \overset{+}{\to})$ *is in* NExpTime *and* PSpace*-complete for tree patterns. The lower bound holds already for containment of unions of tree patterns using only* $\downarrow, \_$.

7 Conclusions

We have shown that under several independent restrictions, CQs have the same complexity of the satisfiability of Boolean combinations, and the containment of unions of queries problem, as tree patterns. Most importantly, vertical tree patterns can be extended with full horizontal CQs over children without increasing the complexity of static analysis tasks. We have also showed that under non-recursive, disjunction-free schemas the complexity of static analysis for CQs without the next-sibling relation is in low levels of the polynomial hierarchy. This could be applied in the analysis of mappings between nested-relational schemas [2]. (We point out the complexity gap for general disjunction-free schemas as an elegant theoretical challenge.) We focused on containment of UCQs, since this is the problem relevant for XML metadata management, but a finer analysis of the containment for CQs would also be desired (the 2ExpTime-lower bound of [9] holds already for validity of CQs). Similarly, patterns with data comparisons might be considered (again, some cases are settled in [9]).

Acknowledgements. This work is part of the *Querying and Managing Navigational Databases* project realized within the Homing Plus programme of the Foundation for Polish Science, co-financed by the European Union from the Regional Development Fund within the Operational Programme Innovative Economy ("Grants for Innovation"). We thank Claire David for inspiring discussions and careful reading of a preliminary version of this paper, and the anonymous referees for helpful comments motivating us to improve the presentation of the paper.

References

1. Amano, S., David, C., Libkin, L., Murlak, F.: On the Tradeoff between Mapping and Querying Power in XML Data Exchange. In: ICDT, pp. 155–164 (2010)
2. Amano, S., Libkin, L., Murlak, F.: XML schema mapping. In: PODS, pp. 33–42 (2009)
3. Arenas, M., Libkin, L.: XML data exchange: consistency and query answering. J. ACM 55(2) (2008)
4. Benedikt, M., Bourhis, P., Senellart, P.: Monadic Datalog Containment. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 79–91. Springer, Heidelberg (2012)
5. Benedikt, M., Fan, W., Geerts, F.: XPath satisfiability in the presence of DTDs. J. ACM 55(2) (2008)
6. Bex, G.J., Neven, F., Van den Bussche, J.: DTDs versus XML Schema: a practical study. In: WebDB, pp. 79–84 (2004)
7. Björklund, H., Gelade, W., Martens, W.: Incremental XPath evaluation. ACM Trans. Database Syst. 35(4), 29 (2010)
8. Björklund, H., Martens, W., Schwentick, T.: Conjunctive Query Containment over Trees. In: Arenas, M. (ed.) DBPL 2007. LNCS, vol. 4797, pp. 66–80. Springer, Heidelberg (2007)
9. Björklund, H., Martens, W., Schwentick, T.: Optimizing Conjunctive Queries over Trees Using Schema Information. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 132–143. Springer, Heidelberg (2008)
10. Bojańczyk, M., Kołodziejczyk, L.A., Murlak, F.: Solutions in XML data exchange. In: ICDT, pp. 102–113 (2011)
11. ten Cate, B., Lutz, C.: The Complexity of Query Containment in Expressive Fragments of XPath 2.0. J. ACM 56(6), 1–48
12. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: STOC, pp. 77–90 (1977)
13. David, C.: Complexity of Data Tree Patterns over XML Documents. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 278–289. Springer, Heidelberg (2008)
14. David, C., Libkin, L., Murlak, F.: Certain answers for XML queries. In: PODS, pp. 191–202 (2010)
15. Fagin, R., Kolaitis, P., Miller, R., Popa, L.: Data exchange: semantics and query answering. Theor. Comp. S. 336, 89–124 (2005)
16. Gottlob, G., Koch, C., Schulz, K.: Conjunctive queries over trees. J. ACM 53, 238–272 (2006)
17. Hidders, J.: Satisfiability of XPath Expressions. In: Lausen, G., Suciu, D. (eds.) DBPL 2003. LNCS, vol. 2921, pp. 21–36. Springer, Heidelberg (2004)
18. Ishihara, Y., Morimoto, T., Shimizu, S., Hashimoto, K., Fujiwara, T.: A Tractable Subclass of DTDs for XPath Satisfiability with Sibling Axes. In: Gardner, P., Geerts, F. (eds.) DBPL 2009. LNCS, vol. 5708, pp. 68–83. Springer, Heidelberg (2009)
19. Lenzerini, M.: Data integration: a theoretical perspective. In: PODS, pp. 233–246 (2002)
20. Marx, M.: XPath with Conditional Axis Relations. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 477–494. Springer, Heidelberg (2004)
21. Miklau, G., Suciu, M.: Containment and equivalence for a fragment of XPath. J. ACM 51(1), 2–45 (2004)

22. Murata, M., Lee, D., Mani, M., Kawaguchi, K.: Taxonomy of XML schema languages using formal language theory. ACM Transactions on Internet Technology 5(4), 1–45 (2005)
23. Neven, F.: Automata Theory for XML Researchers. SIGMOD Record 31(3), 39–46 (2002)
24. Neven, F., Schwentick, T.: On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. Logical Meth. in Comp. Sci. 2(3), 1–30 (2006)
25. Wood, P.T.: Containment for XPath Fragments under DTD Constraints. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 297–311. Springer, Heidelberg (2002)

Reducing a Target Interval to a Few Exact Queries*

Jesper Nederlof[1], Erik Jan van Leeuwen[2], and Ruben van der Zwaan[3]

[1] Utrecht University, The Netherlands
J.Nederlof@uu.nl.
[2] Sapienza University of Rome, Italy
E.J.van.Leeuwen@dis.uniroma1.it.
[3] Maastricht University, The Netherlands
r.vanderzwaan@maastrichtuniversity.nl

Abstract. Many combinatorial problems involving weights can be formulated as a so-called *ranged problem*. That is, their input consists of a universe U, a (succinctly-represented) set family $\mathcal{F} \subseteq 2^U$, a weight function $\omega : U \to \{1, \ldots, N\}$, and integers $0 \leq l \leq u \leq \infty$. Then the problem is to decide whether there is an $X \in \mathcal{F}$ such that $l \leq \sum_{e \in X} \omega(e) \leq u$. Well-known examples of such problems include Knapsack, Subset Sum, Maximum Matching, and Traveling Salesman. In this paper, we develop a generic method to transform a ranged problem into an *exact problem* (i.e. a ranged problem for which $l = u$). We show that our method has several intriguing applications in exact exponential algorithms and parameterized complexity, namely:

- In exact exponential algorithms, we present new insight into whether Subset Sum and Knapsack have efficient algorithms in both time and space. In particular, we show that the time and space complexity of Subset Sum and Knapsack are equivalent up to a small polynomial factor in the input size. We also give an algorithm that solves sparse instances of Knapsack efficiently in terms of space and time.
- In parameterized complexity, we present the first kernelization results on weighted variants of several well-known problems. In particular, we show that weighted variants of Vertex Cover and Dominating Set, Traveling Salesman, and Knapsack all admit polynomial randomized Turing kernels when parameterized by $|U|$.

Curiously, our method relies on a technique more commonly found in approximation algorithms.

1 Introduction

In many computational problems in the field of combinatorial optimization the input partly consists of a set of integers. Since integers are naturally represented

* This work is supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO), project: 'Space and Time Efficient Structural Improvements of Dynamic Programming Algorithms', and by ERC StG project PAAl no. 259515.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 718–727, 2012.

in binary, they can be exponential in the number of bits of the input instance. For many problems this is not an issue, particularly for problems admitting a strongly polynomial-time algorithm (recall that the running time of such an algorithm does not depend on the size of the integers). However, (exponentially) large numbers present a major issue for other problems. For example, in weakly NP-complete problems the large integers are even the sole source of hardness. Strongly NP-complete problems are often studied in their weighted variants, and often such weighted variants are even considerably harder than their unweighted counterpart. In this paper, we present a novel method to reduce the challenges posed by (exponentially) large numbers in the input of NP-complete problems.

We first give a description of the type of problems that we consider. All of the studied problems can be stated according to the following generic pattern. First, there is a universe U (for example the set of vertices or edges of a graph) and a weight function $\omega : U \to \{1, \ldots, N\}$. Second, there is a succinctly-represented set family $\mathcal{F} \subseteq 2^U$. We will assume that membership of the set $\mathcal{F}$ can be determined in polynomial time by an oracle given as part of the input. Finally, we are given two non-negative integers l, u such that $0 \leq l \leq u \leq \infty$. Then the problem is to decide whether there exists an $X \in \mathcal{F}$ such that $\omega(X) \in [l, u]$, where $\omega(X) = \sum_{e \in X} \omega(e)$. We call this a *ranged problem.* If a problem additionally specifies that $l = u$, this is an *exact problem.* We will be mainly interested in the case where N is exponential, or even super-exponential, in $|U|$.

The main question that we consider and answer in this paper is whether the computational complexity of a ranged problem is equal to that of its corresponding exact problem. This question is motivated by the recent availability of powerful tools for exact problems, such as hashing (see e.g. [8]) and interpolation (see e.g. [11,12]), that do not seem directly applicable to ranged problems. Hence we may wonder whether there is a difference between ranged and exact problems from the point of view of computational complexity.

Certain cases of this main question are particularly intriguing. For example, the arguably most fundamental pair of an exact and its corresponding ranged problem is Subset Sum and Knapsack respectively[1]. Recall that in Subset Sum, we are given a set $U = \{1, \ldots, n\}$, a weight function $\omega : U \to \{1, \ldots, N\}$, and an integer $t \leq N$, and we are asked to decide whether there exists an $X \subseteq U$ such that $\omega(X) = t$. In the Knapsack problem we are additionally given a weight function ν and integer b, and we are asked to decide whether there exists a set X with $\omega(X) \geq t$ among all $X \subseteq U$ for which $\nu(X) \leq b$. From the perspective of exact exponential algorithms (see e.g. [7,17] for an introduction), both problems are known to be solvable in $\mathcal{O}^*(2^{n/2})$ time and $\mathcal{O}^*(2^{n/4})$ space [15] (see also [7, Chapter 9]), while the best polynomial-space algorithms are still the trivial brute-force $\mathcal{O}^*(2^n)$-time algorithms. It is an interesting question whether either of these problems can be solved in $\mathcal{O}^*(1.99^n)$ time and polynomial space. Also, are the problems related in the sense that an improved algorithm for Subset Sum would imply an improved algorithm for Knapsack?

[1] In the field of cryptography, "Knapsack" is often used to refer to "Subset Sum".

Another interesting perspective is that of *sparse instances*. It is known that the Subset Sum and Knapsack problems can be solved in pseudo-polynomial time and space using a dynamic programming (DP) algorithm [2]. An intensively studied case of DP is where the DP-table is guaranteed to be sparse. In Subset Sum, for example, this means that the number of distinct sums of the subsets of the given integers is small. Using memorization, this type of sparseness can be easily exploited if we are allowed to use exponential space. Very recently, polynomial-space equivalents of memorization were given in [10] (see also [13, Chapter 6]). The first step in this approach uses hashing, and the second step uses interpolation. It is unclear whether the approach can be extended to Knapsack. One issue is that a good hash function does not hash the target interval to a single interval. Furthermore, interpolation does not apply directly to the ranged case, and the typical solution of adding a few "slack weights" to reduce it to the exact case destroys the sparseness property.

We note that different measures of sparseness for Knapsack have been considered previously. Nemhauser and Üllmann [14] considered the case when the number of Pareto-optimal solutions is small. A solution X for Knapsack is Pareto-optimal if there is no X' with $\nu(X') < \nu(X)$ and $\omega(X') \geq \omega(X)$, or with $\nu(X') \leq \nu(X)$ and $\omega(X') > \omega(X)$. Note that the number of Pareto-optimal solutions is always at most the number of distinct sums in the instance. The algorithm of Nemhauser and Üllmann uses $\mathcal{O}(\sum_{i=k}^{n} p_i)$ time and $\mathcal{O}(\max p_i)$ space to enumerate all Pareto-optimal solutions, where p_i is the number of Pareto-optimal solutions over the first i items. Note that the space requirement is polynomial in the sparseness, whereas the space requirement of the algorithm of [10] is polynomial in the size of the instance. In the framework of smoothed analysis[2], however, the number of Pareto-optimal solutions for Knapsack is polynomial in the instance size [1].

In the field of kernelization (see [4] for a survey), the Subset Sum problem is known to admit a so-called *polynomial randomized kernel* when parameterized by the number of integers [8]. Can a similar kernel be obtained for the Knapsack problem? Again, since [8] heavily relies on hashing, it does not seem to be applicable. Similar questions can be asked for weighted variants of several fundamental problems in the field of kernelization, such as the weighted variant of Vertex Cover. Is there a (randomized Turing) kernel for this problem parameterized by $|U|$, i.e. can we reduce the weights to be at most $2^{|U|^{\mathcal{O}(1)}}$?

Our Results. In this paper, we show that a ranged problem is equally hard as its corresponding exact problem, modulo a factor $\mathcal{O}(|U| \cdot \lg(|U|\, N))$ in the running time. This implies a positive answer on all of the above questions. This result uses a generic and clean method to transform a ranged problem into instances of its corresponding exact problem. The method covers the interval $[l, u]$ with a small number of "fuzzy intervals" such that an integer is in $[l, u]$ if and only if it is in one of the fuzzy intervals. It relies on a scaling technique that is more

[2] Smoothed analysis aims to provide a middle ground between average-case and worst-case analysis. See e.g. [16].

commonly found in approximation algorithms; in fact, the prime example of its use is in the FPTAS for Knapsack [9].

The paper is organized as follows. In Section 2, we introduce the required notation and definitions. In Section 3, we state and prove our main technical contribution. Sections 4 and 5 are dedicated to corollaries of the main theorem in the fields of exact exponential algorithms and kernelization. Finally, we give a conclusion, further remarks, and open questions in Section 6.

2 Preliminaries

Throughout, we use the $\mathcal{O}^*(\cdot)$ notation that suppresses any factor polynomial in the input size of the given problem instance. We use Greek symbols such as ω to denote weight functions, i.e. for a universe U and an integer N, $\omega : U \to \{1, \ldots, N\}$. In this context, we shorthand $\omega(X) = \sum_{e \in X} \omega(e)$ for any $X \subseteq U$. For two integers $l \leq u$, the set of integers $\{l, l+1, \ldots, u\}$ is denoted by $[l, u]$.

A *kernelization algorithm* (or *kernel*) for a parameterized problem Π (that is, a problem together with an input measure k) computes in polynomial time, given an instance (x, k) of Π, a new instance (x', k') of Π such that $(x', k') \in \Pi$ if and only if $(x, k) \in \Pi$, and $|x'| \leq f(k)$ for some computable function f. The instance (x', k') is called a *kernel* of Π, and it is called a *polynomial kernel* if f is a polynomial. Not every problem admits a polynomial kernel, or the polynomial hierarchy collapses to the third level [5]. We refer to [4] for a recent overview.

A generalization of the notion of a kernel is a *Turing kernel*. Here the requirement that given a kernel yields an equivalent instance is relaxed. Instead, a polynomial number of instances with the same size restrictions as before may be produced. Moreover, there should be a polynomial-time algorithm that, given which of the produced instances are a Yes-instance, decides whether the original instance is a Yes-instance. The special case where the algorithm returns the OR of the produced instances is called an OR-kernel or a many-to-one kernel [6]. Generalizing these notions further, we use the adjective "randomized" to indicate that the polynomial-time algorithm computing the final answer may have a constant one-sided error probability. Interestingly, the results of [5] even apply to the randomized variant of the original kernel definition, but not to Turing kernels.

Given a graph $G = (U, E)$, a subset $X \subseteq U$ is a *vertex cover* if $u \in X$ or $v \in X$ for every $(u, v) \in E$, and it is a *dominating set* if $u \in X$ or $(u, v) \in E$ for some $v \in X$ for every $u \in U$. In the weighted Vertex Cover and Dominating Set problems, we are given a graph on vertex set U together with a weight function $\omega : U \to \{1, \ldots, N\}$ and an integer t, and are asked to decide whether there is a vertex cover or dominating set $X \subseteq U$, respectively, such that $\omega(X) \leq t$. A *Hamiltonian cycle* is a subset $X \subseteq E$ such that the graph (V, X) is a cycle. In the Traveling Salesman problem we are given a graph on edge set U together with a weight function $\omega : U \to \{1, \ldots, N\}$ and an integer t, and are asked to decide whether there exists a Hamiltonian cycle $X \subseteq U$ such that $\omega(X) \leq t$.

3 The Main Method

In this section, we give the main technical contribution of the paper: we transform a ranged problem into a small number of exact problems. The naive way to obtain such a transformation would be to return a new instance for each $x \in [l, u]$, thus yielding $u - l$ problems. However, as $u - l$ can be exponential in the size of the input, this procedure is clearly not efficient. Instead, Theorem 1 develops a new family of $\mathcal{O}(|U| \lg(|U|\, N))$ weight functions.

We want to stress that the main conceptual consequence of Theorem 1 is that, modulo a small polynomial factor, weighted subset-selection problems that aim to find a subset of given exact weight are equally hard as those that aim to find a subset with weight in a given interval.

Theorem 1 (Shrinking intervals). *Let U be a set of cardinality n, let $\omega : U \to \{0, \ldots, N\}$ be a weight function, and let $l < u$ be non-negative integers with $u - l > 1$. Then there is a polynomial-time algorithm that returns a set of pairs $\Omega = \{(\omega_1, t_1), \ldots, (\omega_K, t_K)\}$ with $\omega_i : U \to \{0, \ldots, N\}$ and integers $t_1, \ldots, t_K \leq N$ such that*

(C1) K is at most $(5n + 2) \lg(u - l)$, and
(C2) for every set $X \subseteq U$ it holds that $\omega(X) \in [l, u]$ if and only if there exist an index i such that $\omega_i(X) = t_i$.

Proof. The theorem is implemented in Algorithm 1. Let $T(u - l)$ denote the maximum number of pairs that `shrink` returns. We claim that $T(u - l)$ is at most $(5n + 2) \lg(u - l)$. The cases when either l or u is odd can only happen twice in a row before either case one or four occurs. Observe that for the first case our claim is clearly true, and that for the last case we obtain the bound $T(u-l) \leq (5n+2) + T((u-l)/2)$, which clearly meets our claim since $u - l \geq 2$. This settles (C1) and the claim concerning the running time.

We prove (C2) by induction. For the thirst three cases, (C2) clearly holds, so let us directly proceed to the last case. For the forward direction, assume

```
Algorithm shrink($\omega, l, u$)
if $u - l \leq 5n$ then
    return $\{(\omega, l), (\omega, l + 1), \ldots, (\omega, u)\}$.
else if $l$ is odd then
    return $\{(\omega, l)\} \cup$ shrink($\omega, l + 1, u$).
else if $u$ is odd then
    return $\{(\omega, u)\} \cup$ shrink($\omega, l, u - 1$).
else
    For every $e \in U$, set $\omega'(e) = \lfloor \omega(e)/2 \rfloor$.
    $\Omega_l \leftarrow \{(\omega, l), (\omega, l + 1), \ldots, (\omega, l + 3n)\}$.
    $\Omega_r \leftarrow \{(\omega, u), (\omega, u - 1), \ldots, (\omega, u - 2n)\}$.
    return $\Omega_l \cup$ shrink($\omega', (l + 2n)/2, (u - 2n)/2$) $\cup\ \Omega_r$.
```

Algorithm 1. Shrinking intervals

that $\omega(X) \in [l, u]$. If $\omega(X) \in [l, l + 3n] \cup [u - 2n, u]$, then a pair fulfilling (C2) is in $\Omega_l \cup \Omega_r$. So assume that $\omega(X) \in [l + 3n, u - 2n]$. Then a pair fulfilling (C2) will be added in the recursive step by the induction hypothesis, because $\omega'(X) \in [(l + 2n)/2, (u - 2n)/2]$ since

$$\frac{\omega(X) - n}{2} \leq \sum_{e \in X} \left\lfloor \frac{\omega(e)}{2} \right\rfloor = \omega'(X) \leq \omega(X)/2. \tag{1}$$

For the reverse direction, assume that there is a pair $(\omega_i, t_i) \in \Omega$ such that $\omega_i(X) = t_i$. If the pair is from $\Omega_l \cup \Omega_r$, then $\omega(X) \in [l, u]$. So assume that it is added in the recursive step. Then by the induction hypothesis, $\omega'(X) \in [(l + 2n)/2, (u - 2n)/2]$, and $\omega(X) \in [l, u]$ by (1). □

4 Exact Exponential Algorithms

In this section, we demonstrate the applicability of Theorem 1 to exact exponential algorithms. First, we consider the relation between the computational complexity of Knapsack and Subset Sum. It is trivial that any algorithm for Knapsack can be used for Subset Sum: Given an instance (U, ω, b) of Subset Sum, we can define $t = b$ and $\nu(e) = \omega(e)$ for every $e \in U$. Then the instance of Subset Sum is a `Yes`-instance if and only if the constructed instance of Knapsack is a `Yes`-instance. This can be decided by the assumed algorithm for Knapsack. We prove the converse relation below by applying Theorem 1.

Theorem 2. *If there exists an algorithm that decides the* Subset Sum *problem in $\mathcal{O}^*(t(n))$ time and $\mathcal{O}^*(s(n))$ space, then there exists an algorithm that decides the* Knapsack *problem in $\mathcal{O}^*(t(n))$ time and $\mathcal{O}^*(s(n))$ space.*

Proof. Consider a Knapsack instance, consisting of a universe $U = \{1, \ldots, n\}$, weight functions $\omega, \nu : U \to \{1, \ldots, N\}$, and integers b, t. Now we apply Theorem 1 on both weight functions to obtain two sets $\Omega_\omega, \Omega_\nu$. By considering the elements of $\Omega := \Omega_\omega \times \Omega_\nu$ as quadruples, we obtain a set Ω of at most $\mathcal{O}(n^2 \lg^2(nN))$ quadruples $(\omega_i, \nu_i, b_i, t_i)$ such that for every $X \subseteq U$ it holds that $\nu(X) \in [0, b]$ and $\omega(X) \in [t, nN]$ if and only if there exists an i such that $\nu_i(X) = b_i$ and $\omega_i(X) = t_i$. It remains to show that, for every quadruple $(\omega_i, \nu_i, b_i, t_i)$, we can determine whether there exists an $X \subseteq U$ such that $\nu_i(X) = b_i$ and $\omega(X) = t_i$. To do this, we create an instance of Subset Sum by concatenating the integer values. Specifically, given a quadruple $(\omega_i, \nu_i, b_i, t_i)$, define

$$\alpha_i(e) = \nu_i(e)N(n + 1) + \omega_i(e) \text{ for every } e \in U \qquad \text{and} \qquad c_i = b_i(n + 1)N + t_i.$$

It is easy to see that, since $\omega_i(X) \leq N(n+1)$, $\alpha_i(X) = c_i$ if and only if $\nu_i(X) = b_i$ and $\omega(X) = t_i$. Then the assumed algorithm for Subset Sum can be used to decide for every quadruple $(\omega_i, \nu_i, b_i, t_i)$ whether there exists $X \subseteq U$ such that $\alpha_i(X) = c_i$. This in turn enables us to decide the Knapsack instance. The bound on the time and space complexity follows immediately from the fact that $|\Omega|$ is $\mathcal{O}(n^2 \lg^2(nN))$, which is polynomial in the size of the instance. □

As an easy corollary, we observe that we can apply binary search to even deal with the maximization variant of Knapsack.

Corollary 1. *There exists an algorithm that decides the* Subset Sum *problem in $\mathcal{O}^*(t(n))$ time and $\mathcal{O}^*(s(n))$ space if and only if there exists an algorithm that solves the* Maximum Knapsack *problem in $\mathcal{O}^*(t(n))$ time and $\mathcal{O}^*(s(n))$ space.*

We can use the ideas in the proof of Theorem 2 to give another result on Knapsack. To this end, we assume that the given instance of Knapsack or Subset Sum is sparse, that is, the number of distinct sums in the instance is small. We recall a recent result of Kaski et al. [10].

Theorem 3 ([10]). *There is an algorithm that decides an instance (U, ω, t) of* Subset Sum *in $\mathcal{O}^*(S)$ expected time and $\mathcal{O}^*(1)$ space, where $S = |\{\omega(X) : X \subseteq U\}|$.*

Using Theorem 1 and the ideas of Theorem 2, we can prove the following.

Theorem 4. *There is an algorithm that decides an instance (U, ω, ν, t, b) of* Knapsack *in $\mathcal{O}^*(S)$ expected time and $\mathcal{O}^*(1)$ space, where $S = |\{(\omega(X), \nu(X)) : X \subseteq U\}|$.*

To prove Theorem 4, we require the following auxiliary lemma.

Lemma 1. *Let U be a set of n elements, let $\omega, \nu : U \to \{1, \ldots, N\}$ be weight functions, let $p \geq 1$ be an integer, and let $\tilde{\omega}(e) = \lfloor \omega(e)/p \rfloor$ and $\tilde{\nu}(e) = \lfloor \nu(e)/p \rfloor$ for every $e \in U$. Then*

$$|\{(\tilde{\omega}(X), \tilde{\nu}(X)) : X \subseteq U\}| \leq n^2 \cdot |\{(\omega(X), \nu(X)) : X \subseteq U\}|.$$

Proof. For any pair of integers (x, y), consider the set $\mathcal{Z} = \{X \subseteq U : \omega(X) = x, \nu(X) = y\}$. For each $X \in \mathcal{Z}$, we have that $\omega(X) - n \leq p \cdot \tilde{\omega}(X) \leq \omega(X)$ and $\nu(X) - n \leq p \cdot \tilde{\nu}(X) \leq \nu(X)$. Therefore, $|\{(\tilde{\omega}(X), \tilde{\nu}(X)) : X \in \mathcal{Z}\}| \leq n^2$, and the lemma follows. □

Proof (of Theorem 4). We first apply the same construction as in the proof of Theorem 2 to obtain pairs (α_i, c_i). We then apply the algorithm of Theorem 3 on all of these pairs and return `Yes` if the algorithm finds an index i and a set $X \subseteq U$ such that $\alpha_i(X) = c_i$.

It remains to prove that this introduces at most a polynomial overhead. Since the number of pairs is bounded by a polynomial in the input length, it suffices to show that for every i, the quantity $|\{\alpha_i(X) : X \subseteq U\}|$ is at most $\mathcal{O}^*(S)$. Observe that new weight functions are created in two places. First when Theorem 1 is invoked: note that in Algorithm 1, all created weight functions are effectively obtained by halving the weights x times and rounding down, which is equivalent to truncating the bitstring or dividing by 2^x and rounding down. Hence, for all created weight functions ω_i, we have that $|\{(\omega_i(X), \nu_i(X)) : X \subseteq U\}| \leq n^2 \cdot S$ for every i by Lemma 1. The second place is when α is defined by concatenating the integers: then $|\{\alpha_i(X) : X \subseteq U\}| = |\{(\omega_i(X), \nu_i(X)) : X \subseteq U\}| \leq n^2 \cdot S$. Hence the overhead is at most polynomial. □

5 Kernelization

In this section we show that Theorem 1 can be used in combination with a known kernelization technique to reduce the number of bits needed to represent the weights of weighted minimization problems to an amount that is polynomial in the number of bits needed to represent the remainder of the input instance.

Theorem 5. *The weighted variants of the* Vertex Cover *and* Dominating Set *problems,* Traveling Salesman*, and* Knapsack *all admit polynomial randomized Turing kernels when parameterized by* $|U|$*.*

We need the following lemma from Harnik and Naor [8], which uses randomization to reduce the weights.

Lemma 2 ([8]). *Let U be a set of size n. There exists a polynomial-time algorithm that, given $\omega : U \to \{0, \dots, N\}$, an integer t, and a real $\epsilon > 0$, returns $\omega' : U \to \{0, \dots, M\}$ and integers $t_1, \dots, t_n \leq M$ where $M \leq 2^n \cdot \mathrm{poly}(n, \lg N, \epsilon^{-1})$ such that for every set family $\mathcal{F} \subseteq 2^U$:*

(R1) if there is an $X \in \mathcal{F}$ such that $\omega(X) = t$, then there exist i such that $\omega'(X) = t_i$,
(R2) if there is no $X \in \mathcal{F}$ such that $\omega(X) = t$ then

$$\mathrm{Prob}[\textit{there exist } i \textit{ and } X \in \mathcal{F} \textit{ such that } \omega'(X) = t_i] \leq \mathcal{O}(\epsilon). \tag{2}$$

We give the proof of the lemma in the appendix for completeness. It relies on the fact that in every interval of length l the number of primes is roughly $l/\ln l$ and that a random prime can be constructed in time polylogarithmic in the upper bound of the interval.

Proof (of Theorem 5). In all problems mentioned in the statement of Theorem 5, there is a set family $\mathcal{F} \subseteq 2^U$ and we are asked whether there exists an $X \in \mathcal{F}$ such that either $\omega(X) \in [0, t]$ or $\omega(X) \in [t, nN]$ (depending on the problem). Note that we can assume that $\lg N \leq 2^{|U|}$; otherwise the input is of size at least $2^{|U|}$ and we can use a trivial brute-force algorithm to solve the instance and reduce it to an equivalent instance of constant size.

Now we use Theorem 1 to obtain a set Ω of $\ell = \mathcal{O}(n \lg(nN))$ pairs (ω_i, t_i) and reduce the original problem to detecting whether there exists a pair $(\omega_i, t_i) \in \Omega$ and $X \in \mathcal{F}$ such that $\omega_i(X) = t_i$. To reduce the latter problem further, we apply the algorithm of Lemma 2, setting $\epsilon = \epsilon'/\ell$ for some small value of ϵ'. Hence, for every (ω_i, t_i), we obtain a weight function ω'_i and n integers $t'_{i1}, \dots, t'_{in}$ such that if there exists an $X \in \mathcal{F}$ with $\omega_i(X) = t_i$, then there exists a j such that $\omega'_i(X) = t'_{ij}$ for some j and otherwise (2) holds. Hence, this procedure generates $\mathcal{O}(n^2 \lg(nN))$ pairs such that (i) if there is $X \in \mathcal{F}$ with $\omega(X) \in [0, t]$, a pair (ω'_i, t'_{ij}) with $\omega'_i(X) = t'_{ij}$ is generated (ii) if there is no $X \in \mathcal{F}$ with $\omega(X) \in [0, t]$:

$$\mathrm{Prob}[\text{there exist } i, j \text{ and } X \in \mathcal{F} \text{ such that } \omega'_i(X) = t'_{ij}] \leq \ell \cdot \epsilon = \mathcal{O}(\epsilon').$$

Now we have reduced the original decision problem to a problem that is clearly in NP: indeed, we can obtain the correct $X \subseteq U$ in non-deterministic polynomial time and verify whether it satisfies $X \in \mathcal{F}$ (that is, is it a vertex cover, dominating set, ...) and $\omega'_i(X) = t'_{ij}$. Then, since the original problem (Vertex Cover, Dominating Set, ...) is NP-complete, we can reduce the problem to instances of the original problem with a Karp-reduction. Hence we have reduced one problem instance to many problem instances such that:

- if the original instance is a Yes-instance, then one of the created instances is also a Yes-instance;
- if the original instance is a No-instance, then with constant probability all created instances are No-instances.

Thus it remains to show that the number and the description lengths of the created instances are bounded by a polynomial in the original input size. To see that this is the case, first note that after applying Lemma 2 we have $\mathcal{O}(n^2 \lg(nN))$ pairs of weight functions bounded by $2^n \cdot \text{poly}(n, \lg N, \epsilon^{-1})$. Since we assumed that $\lg N \leq 2^n$, these weight functions are represented by polynomially many bits. Then, the theorem follows from the fact that a Karp-reduction increases the size of a problem by at most a polynomial factor. □

6 Conclusion

We presented a generic and simple method to convert ranged problems into exact problems. While this result is already interesting by itself given its generality, we also gave a number of corollaries that followed by combining our method with techniques for exact problems already available from previous work. It is worth emphasizing the generality of our results in Section 3 and Section 4. For example, in the context of exact exponential algorithms, Traveling Salesman seems to be significantly harder than its unweighted version, Hamiltonian Cycle. Recently, the latter was shown to be solvable in $\mathcal{O}^*(1.66^n)$ time and polynomial space [3], whereas the best algorithm for Traveling Salesman that is insensitive to large weights uses $\mathcal{O}^*(2^n)$ and space. By combining the hashing idea of [10] with our method, it is for example possible to obtain a polynomial-space algorithm for Traveling Salesman that runs in $\mathcal{O}^*(2^n W)$ time, where $W = |\{\omega(X) : X \text{ is a Hamiltonian cycle.}\}|$.

We leave the reader with several interesting open questions:

- Can we get a "classical"(i.e. non-Turing or many-to-one) polynomial kernel for the considered parameterized version of weighted Vertex Cover?
- Further, significant reduction of the weights in polynomial time seems hard (for example, it would imply an improved pseudo-polynomial algorithm for Subset Sum), but is it possible in pseudo-polynomial time for example for Traveling Salesman?
- When is minimizing/maximizing as hard as the general range problem?
- Can we modify Theorem 1 to make it counting-preserving? More precisely, can we obtain a variant of the theorem with a third condition $(C3)$ saying that for every $X \subseteq U$ there is at most one i such that $\omega_i(X) = t_i$?

References

1. Beier, R., Vöcking, B.: Random knapsack in expected polynomial time. J. Comput. Syst. Sci. 69(3), 306–329 (2004)
2. Bellman, R.E.: Dynamic Programming (reprint 2003). Dover Publications, Incorporated (1954)
3. Björklund, A.: Determinant sums for undirected Hamiltonicity. In: FOCS, pp. 173–182. IEEE Computer Society (2010)
4. Bodlaender, H.L.: Kernelization: New Upper and Lower Bound Techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)
5. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. 75(8), 423–434 (2009)
6. Fernau, H., Fomin, F.V., Lokshtanov, D., Raible, D., Saurabh, S., Villanger, Y.: Kernel(s) for problems with no kernel: On out-trees with many leaves. In: Albers, S., Marion, J.-Y. (eds.) STACS. LIPIcs, vol. 3, pp. 421–432. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2009)
7. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms, 1st edn. Springer, New York (2010)
8. Harnik, D., Naor, M.: On the compressibility of *np* instances and cryptographic applications. SIAM J. Comput. 39(5), 1667–1713 (2010)
9. Ibarra, O., Kim, C.: Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM 22, 463–468 (1975)
10. Kaski, P., Koivisto, M., Nederlof, J.: Homomorphic hashing for sparse coefficient extraction. Manuscript (2012)
11. Lokshtanov, D., Nederlof, J.: Saving space by algebraization. In: Schulman, L.J. (ed.) STOC, pp. 321–330. ACM (2010)
12. Mansour, Y.: Randomized interpolation and approximation of sparse polynomials. SIAM J. Comput. 24(2), 357–368 (1995)
13. Nederlof, J.: Space and Time Efficient Structural Improvements of Dynamic Programming Algorithms. PhD thesis, University of Bergen (2011)
14. Nemhauser, G.L., Ullmann, Z.: Discrete dynamic programming and capital allocation. Management Science 15(9), 494–505 (1969)
15. Schroeppel, R., Shamir, A.: A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. SIAM J. Comput. 10(3), 456–464 (1981)
16. Spielman, D.A., Teng, S.-H.: Smoothed analysis: an attempt to explain the behavior of algorithms in practice. Commun. ACM 52(10), 76–84 (2009)
17. Woeginger, G.J.: Open problems around exact algorithms. Discrete Applied Mathematics 156(3), 397–405 (2008)

Maximum Cliques in Graphs with Small Intersection Number and Random Intersection Graphs

Sotiris Nikoletseas[1,2,⋆], Christoforos Raptopoulos[1,⋆⋆], and Paul G. Spirakis[1,2,⋆⋆⋆]

[1] Computer Technology Institute and Press "Diophantus",
P.O. Box 1382, 26504 Patras, Greece
[2] University of Patras, 26500 Patras, Greece
nikole@cti.gr, raptopox@ceid.upatras.gr, spirakis@cti.gr

Abstract. In this paper, we relate the problem of finding a maximum clique to the intersection number of the input graph (i.e. the minimum number of cliques needed to edge cover the graph). In particular, we consider the maximum clique problem for graphs with small intersection number and random intersection graphs (a model in which each one of m labels is chosen independently with probability p by each one of n vertices, and there are edges between any vertices with overlaps in the labels chosen).

We first present a simple algorithm which, on input G finds a maximum clique in $O(2^{2^m+O(m)} + n^2 \min\{2^m, n\})$ time steps, where m is an upper bound on the intersection number and n is the number of vertices. Consequently, when $m \leq \ln\ln n$ the running time of this algorithm is polynomial.

We then consider random instances of the random intersection graphs model as input graphs. As our main contribution, we prove that, when the number of labels is not too large ($m = n^\alpha, 0 < \alpha < 1$), we can use the label choices of the vertices to find a maximum clique in polynomial time whp. The proof of correctness for this algorithm relies on our Single Label Clique Theorem, which roughly states that whp a "large enough" clique cannot be formed by more than one label. This theorem generalizes and strengthens other related results in the state of the art, but also broadens the range of values considered (see e.g. [22] and [4]).

As an important consequence of our Single Label Clique Theorem, we prove that the problem of inferring the complete information of label choices for each vertex from the resulting random intersection graph (i.e. the *label representation of the graph*) is *solvable* whp; namely, the maximum likelihood estimation method will provide a unique solution (up to permutations of the labels). Finding efficient algorithms for constructing such a label representation is left as an interesting open problem for future research.

[⋆] Research partially supported by EU/FIRE HOBNET project - STREP ICT-257466.

[⋆⋆] Research supported by SHARPEN - PE6(1081), funded by the European Union and the Greek state under the NSRF Action "Strengthening postdoctoral researchers".

[⋆⋆⋆] Research partially supported by RIMACO project - ERC/StG - 210743.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 728–739, 2012.

1 Introduction

A *clique* in an undirected graph G is a subset of vertices any two of which are connected by an edge. The cardinality of the maximum clique is called the *clique number* of G. The problem of finding the maximum clique in an arbitrary graph is fundamental in Theoretical Computer Science and appears in many different settings. As an example, consider a social network where vertices represent people and edges represent mutual acquaintance. Finding a maximum clique in this network corresponds to finding the largest subset of people who all know each other. More generally, the analysis of large networks in order to identify communities, clusters, and other latent structure has come to the forefront of much research. The Internet, social networks, bibliographic databases, energy distribution networks, and global networks of economies are some of the examples motivating the development of the field.

It is well known that determining the clique number of an arbitrary graph is NP-complete [16]. In fact, the fastest algorithm known today runs in time $O(1.1888^n)$ [20], where n is the number of vertices in the graph. Moreover, the best known approximation algorithm for the clique number has a performance guarantee of $O\left(\frac{n(\log\log n)^2}{(\log n)^3}\right)$ [8] (there are algorithms with better approximation ratios for graphs with large clique number; see e.g. [1]). Even though this approximation ratio appears to be weak at first glance, there are several results on hardness of approximation which suggest that there can be no approximation algorithm with an approximation ratio significantly less than linear (see e.g. [11]). It was also shown in [5] that, if k is the clique number, then the clique problem cannot be solved in time $n^{o(k)}$, unless the exponential time hypothesis fails (note that the brute force search algorithm runs in time $O(n^k k^2)$, which seems quite close).

The intractability of the maximum clique problem for arbitrary graphs lead researchers to the study of the problem for appropriately generated random graphs. In particular, for Erdős-Rényi random graphs $G_{n,\frac{1}{2}}$ (i.e. random graphs in which each edge appears independently with probability $\frac{1}{2}$), there are several greedy algorithms that find a clique of size about $\ln n$ with high probability (whp, i.e. with probability that tends to 1 as n goes to infinity), see e.g. [10,15]. Since the clique number of $G_{n,\frac{1}{2}}$ is asymptotically equal to $2\ln n$ with high probability, these algorithms approximate the clique number by a factor of 2. In fact, it was conjectured that finding a clique of size $(1+\epsilon)\ln n$ (for a constant $\epsilon > 0$), with probability at least $\frac{1}{2}$, would require techniques beyond the current limits of complexity theory. This belief was strengthened by the fact that the Metropolis algorithm also fails to find the maximum clique in $G_{n,\frac{1}{2}}$ (see [12]). A more dramatized version of the above conjecture was presented in [12], stating that the problem of finding an $1.01\ln n$ clique remains hard even if the input graph is a $G_{n,\frac{1}{2}}$ random graph in which we have planted a randomly chosen clique of size $n^{0.49}$. This conjecture has some interesting cryptographic consequences, as shown in [13]. It also seems tight, since finding the maximum clique in the case where the planted clique has size at least $\sqrt{n}$ can be done in polynomial time by using spectral properties of the adjacency matrix of the graph (see [2]). We finally note that there are quite a few nice results concerning generalizations of the planted clique problem in various (quite general) random graphs models (see e.g. [6,7]).

1.1 Our Contribution

In this work, we complement the state of the art by relating the maximum clique problem to the intersection number of the input graph G (i.e. the minimum number of cliques that can edge cover G). In particular, we consider the maximum clique problem for graphs with small intersection number and random intersection graphs.

More analytically, we begin by considering arbitrary graphs with small intersection number. We present a simple algorithm which, on input G finds a maximum clique in $O(2^{2^m+O(m)} + n^2 \min\{2^m, n\})$ time steps, where m is an upper bound on the intersection number of G and n is the number of vertices. Consequently, when $m \leq \ln\ln n$ the running time of this algorithm is polynomial. We note here that computing the exact value of the independence number of G is itself an NP-complete problem, but this knowledge is only needed in the analysis of the algorithm.

We then consider random instances of the random intersection graphs model (introduced in [14,22]) as input graphs. In this model, denoted by $\mathcal{G}_{n,m,p}$, each one of m labels is chosen independently with probability p by each one of n vertices, and there are edges between any vertices with overlaps in the labels chosen. Random intersection graphs are relevant to and capture quite nicely social networking. Indeed, a social network is a structure made of nodes (individuals or organizations) tied by one or more specific types of interdependency, such as values, visions, financial exchange, friends, conflicts, web links etc. Social network analysis views social relationships in terms of nodes and ties. Nodes are the individual actors within the networks and ties are the relationships between the actors. Other applications include oblivious resource sharing in a (general) distributed setting, efficient and secure communication in sensor networks [17], interactions of mobile agents traversing the web etc. Even epidemiological phenomena (like spread of disease) tend to be more accurately captured by this "interaction-sensitive" random graph model [3].

As our main contribution, we prove that, when the number of labels is not too large, we can use the label choices of the vertices to find a maximum clique in polynomial time (in the number of labels m and vertices n of the graph). Most of the work in this paper is devoted in proving our Single Label Clique Theorem (Theorem 3 in Section 4). The theorem states that when the number of labels is less than the number of vertices, any large enough clique in a random instance of $\mathcal{G}_{n,m,p}$ is formed by a single label. This statement may seem obvious when p is small, but it is hard to imagine that it still holds for *all* "interesting" values for p (see also the discussion in Section 2). Indeed, when $p = o\left(\sqrt{\frac{1}{nm}}\right)$, by slightly modifying an argument of [4], we can see that $G_{n,m,p}$ almost surely has no cycle of size $k \geq 3$ whose edges are formed by k distinct labels (alternatively, the intersection graph produced by reversing the roles of labels and vertices is a tree). On the other hand, for larger p a random instance of $\mathcal{G}_{n,m,p}$ is far from perfect[1] and the techniques of [4] do not apply (for a more thorough discussion see the beginning of Section 4). By using the Single Label Clique Theorem, we provide a tight bound on the clique number of $G_{n,m,p}$ when $m = n^{\alpha}, \alpha < 1$. A lower bound in the special case where mp^2 is constant, was given in [22]. We considerably broaden this range of values to also include vanishing values for mp^2 and also provide an asymptotically tight upper bound.

We claim that our proof also applies for $\alpha < 2$, provided p is not too small. We should note here that in [9] the authors prove the equivalence (measured in terms of total variation distance) of random intersection graphs and Erdős-Rényi random graphs, when $m = n^{\alpha}, \alpha > 6$. This bound on the number of labels was improved in [21], by showing equivalence of sharp threshold functions among the two models for $\alpha \geq 3$. In view of these results, we expect that our work will shed light also in the problem of finding maximum cliques in Erdős-Rényi random graphs.

Finally, as yet another consequence of our Single Label Clique Theorem, we prove that the problem of inferring the complete information of label choices for each vertex from the resulting random intersection graph (i.e. the *label representation of the graph*) is *solvable* whp; namely, the maximum likelihood estimation method will provide a unique solution (up to permutations of the labels).[2] In particular, given values m, n and p, such that $m = n^{\alpha}, 0 < \alpha < 1$, and given a random instance of the $\mathcal{G}_{n,m,p}$ model, the label choices for each vertex are uniquely defined. Finding efficient algorithms for constructing such a label representation is left as an open problem for future research.

1.2 Organization of the Paper

In Section 2 we formally define random intersection graphs. We also provide some useful definitions and notation which are used throughout the paper. The relation of the intersection number to the clique number of an arbitrary graph is discussed in Section 3. Section 4 is devoted to the proof of our Single Label Clique Theorem for random intersection graphs. The consequences of our main theorem concerning the efficient construction of a maximum clique and the uniqueness of the label representation of $G_{n,m,p}$ are presented in Section 5. Finally, we discuss the presented results and further research in Section 6.

[1] A *perfect graph* is a graph in which the chromatic number of every induced subgraph equals the size of the largest clique of that subgraph. Consequently, the clique number of a perfect graph is equal to its chromatic number.

[2] More precisely, if $\mathcal{B}$ is the set of different label choices that can give rise to a graph G, then the problem of inferring the complete information of label choices from G is *solvable* if there is some $B^* \in \mathcal{B}$ such that $\Pr(B^*|G) > \Pr(B|G)$, for all $\mathcal{B} \ni B \neq B^*$.

2 Definitions and Preliminaries

The formal definition of the random intersection graphs model is as follows:

Definition 1 (Random Intersection Graph - $\mathcal{G}_{n,m,p}$ [14,22]). *Consider a universe $\mathcal{M} = \{1, 2, \ldots, m\}$ of elements and a set of n vertices V. Assign independently to each vertex $v \in V$ a subset S_v of $\mathcal{M}$, choosing each element $i \in \mathcal{M}$ independently with probability p and draw an edge between two vertices $v \neq u$ if and only if $S_v \cap S_u \neq \emptyset$. The resulting graph is an instance $G_{n,m,p}$ of the random intersection graphs model.*

In this model we also denote by L_i the set of vertices that have chosen label $i \in M$. Given $G_{n,m,p}$, we will refer to $\{L_i, i \in \mathcal{M}\}$ as its *label representation.* Consider the bipartite graph with vertex set $V \cup \mathcal{M}$ and edge set $\{(v, i) : i \in S_v\} = \{(v, i) : v \in L_i\}$. We will refer to this graph as the *bipartite random graph $B_{n,m,p}$ associated to $G_{n,m,p}$*. Notice that the associated bipartite graph is uniquely defined by the label representation.

It follows from the definition of the model that the edges in $G_{n,m,p}$ are not independent. In particular, the (unconditioned) probability that a specific edge exists is $1 - (1 - p^2)^m$. Therefore, if mp^2 goes to infinity with n, then this probability goes to 1. In the paper, we will thus consider the "interesting" range of values $mp^2 = O(1)$ (i.e. the range of values for which the unconditioned probability that an edge exists does not go to 1). Furthermore, as is usual in the literature, we will assume that the number of labels is some power of the number of vertices, i.e. $m = n^\alpha$, for some $\alpha > 0$.

The following definition will also be useful:

Definition 2 (Intersection number). *The intersection number of a graph G is the smallest number of cliques needed to cover all of the edges of G.*

2.1 Notation

We use the convention that the random intersection graphs model is denoted by $\mathcal{G}_{n,m,p}$ (i.e. with a calligraph $\mathcal{G}$), while a specific random instance of the model is denoted by $G_{n,m,p}$ (i.e. with a simple G).

For a vertex $v \in V$, we denote by $N_G(v)$ the set of neighbors of v in G. We will say that two vertices $v, u \in V$ belong to the same *closed neighborhood in G* and we will write $v \leftrightarrow_G u$ if and only if $N_G(v) \cup \{v\} = N_G(u) \cup \{u\}$.

Let $\mathcal{C}'$ denote a partition of the vertex set V of a graph G and let $v \in V$. We will denote by $\mathcal{C}'[v]$ the unique set inside $\mathcal{C}'$ that contains v, that is $\mathcal{C}'[v] = \{C' \in \mathcal{C}' : v \in C'\}$.

Throughout the paper, the notation $f(n) \sim g(n)$ means that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$ or equivalently $f(n) = g(n) + o(g(n))$.

3 An Algorithm for Maximum Clique

In this section we consider arbitrary graphs as input graphs for the maximum clique problem. In particular, we relate the running time of the following algorithm to the intersection number of the input graph G. The key observation behind the design of Algorithm FIND_MAX-CLIQUE is that a closed neighborhood is either a proper subset of or disjoint from a maximum clique. Therefore, we can reduce every closed neighborhood to a single vertex and then search for a clique in the reduced graph G' that corresponds to a maximum clique in G.

```
Algorithm. FIND_MAX-CLIQUE
Input: $G = (V, E)$
1. Set $U = V$ and $\mathcal{C}' = \emptyset$;
2. while $U \neq \emptyset$ do
3.     Pick $v \in U$ and let $C' = \{u \in U : u \leftrightarrow_G v\}$;
4.     Include $C'$ in $\mathcal{C}'$;
5.     Set $U = U \backslash C'$; endwhile
6. Let $G' = (V', E')$ be an induced subgraph of $G$ that has exactly one vertex
   for every set $C' \in \mathcal{C}'$;
7. Using exhaustive search, find a clique $S$ in $G'$ such that $| \cup_{v' \in S} \mathcal{C}'[v']|$ is
   maximum;
8. Output $Q = \cup_{v' \in S} \mathcal{C}'[v']$;
```

An example of a graph G and corresponding G' (as constructed in step 6) can be found in the full version of the paper [18].

3.1 Analysis of FIND_MAX-CLIQUE

We first present the following lemma that concerns basic properties of the relation $\leftrightarrow_G$. Its easy proof can be found in the full version of the paper [18].

Lemma 1. *The closed neighborhood relation $\leftrightarrow_G$ is an equivalence relation with the following properties:*

1. *It is an equivalence relation which partitions the vertex set V in equivalence classes called closed neighborhoods.*
2. *A closed neighborhood is a clique. Two closed neighborhoods either form a clique, or no edge between their vertices exists.*

The following theorem concerns the correctness of the Algorithm FIND_MAX-CLIQUE. Its proof can be found in the complete version of the paper [18].

Theorem 1 (Correctness). *FIND_MAX-CLIQUE correctly outputs a maximum clique in G.*

The following result relates the running time of Algorithm FIND_MAX-CLIQUE to the intersection number of its input graph G. Its proof can be found in the complete version of the paper [18].

Theorem 2 (Efficiency). *Let $G = (V, E)$ be a graph with intersection number m. Then FIND_MAX-CLIQUE on input G finds a maximum clique in $O(2^{2^m + O(m)} + n^2 \min\{2^m, n\})$ time steps.*

Note that the algorithm does not need the actual value of the independence number. We only use this information for bounding its running time. The following is a direct consequence of Theorem 2.

Corollary 1. *Let $m \leq \ln \ln n$ be an upper bound on the independence number of an arbitrary undirected graph G on n vertices. Then there is an algorithm that finds the maximum clique of G in time $O(n^2 \ln n)$.*

As a final remark, since the intersection number of $G_{n,m,p}$ is at most m (but could be even less), the above result also holds for any random instance of the random intersection graphs model with at most $\ln \ln n$ labels.

4 Clique Number for $m = n^{\alpha}, 0 < \alpha < 1$

In this section we give a tight bound on the clique number of $G_{n,m,p}$ when $m = n^{\alpha}, \alpha < 1$. A lower bound in the special case where mp^2 is constant, was given in [22]. We considerably broaden this range of values to also include vanishing values for mp^2 and also provide a tight upper bound.

We will also assume, without loss of generality, that $p = \Omega\left(\sqrt{\frac{1}{nm}}\right)$. Indeed, when $p = o\left(\sqrt{\frac{1}{nm}}\right)$, by slightly modifying an argument of [4], we can see that $G_{n,m,p}$ almost surely has no cycle of size $k \geq 3$ whose edges are formed by k distinct labels. Therefore, the maximum clique of $G_{n,m,p}$ when $p = o\left(\sqrt{\frac{1}{nm}}\right)$, is formed by exactly one label. As a matter of fact, if L_i is the set of vertices that have chosen label $i \in \mathcal{M}$, then the maximum clique is equal to L_l, where $l \in \arg\max_{i \in \mathcal{M}} |L_i|$. Furthermore, since $G_{n,m,p}$ is chordal whp (see Lemma 5 in [4]), the maximum clique can be found in polynomial time.

We stress out the fact that the techniques employed to provide the algorithmic and structural results in [4] cannot be used in the case where $p = \Omega\left(\sqrt{\frac{1}{nm}}\right)$. In particular, $G_{n,m,p}$ is far from perfect, especially in the the case $mp = \omega(\ln n)$ (which is included in the range of values that we study here). An intuitive justification is as follows: when $mp = \omega(\ln n)$, then the size of the label sets of every vertex are highly concentrated around their mean value mp. Therefore, the statistical behavior of $G_{n,m,p}$ is expected to be similar to the statistical behavior of uniform random intersection graphs $G_{n,m,\lambda}$, in which each vertex selects exactly $\lambda = mp$ labels from $\mathcal{M}$. It was proved in [19] (part (iii) in Corollary 2), that the size of the maximum independent set when $m = n^{\alpha}, \alpha < 1$ and $\lambda = \omega(\ln n)$, is asymptotically equal to $2(1-\alpha)\frac{m \ln n}{\lambda^2}$. Therefore, when $mp = \omega(\ln n)$, the size of the maximum independent set in $G_{n,m,p}$ will be around $\Theta\left(\frac{\ln n}{mp^2}\right)$, so its chromatic number will be $\Omega\left(\frac{nmp^2}{\ln n}\right)$. However, as can be seen in Corollary 4 (which is a direct consequence of our main theorem), the size of the maximum clique in $G_{n,m,p}$ when $m = n^{\alpha}, \alpha < 1$ and $mp^2 = O(1)$ is asymptotically equal to np. This is much smaller than the lower bound $\Omega\left(\frac{nmp^2}{\ln n}\right)$ on the chromatic number in the case $mp = \omega(\ln n)$. Therefore, $G_{n,m,p}$ is far from perfect in this range of values.

We first provide some concentration results concerning the number of vertices that have chosen a particular label and the number of labels that have been chosen by a particular vertex. The proof of the following lemma can be found in the full version of the paper [18].

Lemma 2. *Let $G_{n,m,p}$ be a random instance of the random intersection graphs model with $m = n^\alpha, 0 < \alpha < 1$ and $p = \Omega\left(\sqrt{\frac{1}{nm}}\right)$. Then the following hold:*

A. *Let L_i be the set of vertices that have chosen label $i \in \mathcal{M}$. Then* $\Pr(\exists i \in \mathcal{M} : ||L_i| - np| \geq 3\sqrt{np \ln n}) \leq \frac{1}{n^3} \to 0$.
B. *Let also S_v denote the set of labels that were chosen by vertex v. Then* $\Pr(\exists v \in V : |S_v| > mp + 3\sqrt{mp \ln m} + \ln n) \to 0$.

Notice that the above lemma provides a lower bound on the clique number. However, a clique in $G_{n,m,p}$ can be formed by combining more than one label. Clearly, a clique Q which is not formed by a single label will need at least 3 labels, since 2 labels cannot cover all the edges needed for Q to be a clique. In the discussion below, we will provide a much larger lower bound on the number of labels needed to form a clique Q of size $|Q| \sim np$ which is not formed by a single label. The following definition will be useful.

Definition 3. *Denote by $A_{y,x}$ the event that there are two disjoint sets of vertices $V_1, V_2 \subset V$, where $|V_1| = y$ and $|V_2| = x$ such that the following hold:*

1. *All vertices in V_1 have chosen some label l_0, i.e. $l_0 \in \cap_{u \in V_1} S_u$.*
2. *None of the vertices in V_2 has chosen l_0, i.e. $l_0 \notin \cup_{v \in V_2} S_v$.*
3. *Every vertex in V_1 is connected to every vertex in V_2.*

As a warm-up, we present the following technical lemma, which is a first indication that in a $G_{n,m,p}$ graph, whp we cannot have y too large and x too small at the same time. This lemma is also used as a starting step in the proof of our main theorem. Its proof can be found in the full version of the paper [18].

Lemma 3. *Let $G_{n,m,p}$ be a random instance of the random intersection graphs model with $m = n^\alpha, 0 < \alpha < 1$ and $p = \Omega\left(\sqrt{\frac{1}{nm}}\right)$ and $mp^2 = O(1)$. Then, for any $y \geq np\left(1 - o\left(\frac{1}{\ln n}\right)\right)$,* $\Pr(A_{y,1}) = o(1)$.

The above lemma has the following useful, alternative interpretation (for the proof see the full version of the paper [18]):

Corollary 2. *Let $G_{n,m,p}$ be a random instance of the random intersection graphs model with $m = n^\alpha, 0 < \alpha < 1$, $p = \Omega\left(\sqrt{\frac{1}{nm}}\right)$ and $mp^2 = O(1)$. Let also Q be a clique in $G_{n,m,p}$ that is not formed by a single label and also $|Q| \sim np$. If $l_0 \in \mathcal{M}$ is any label chosen by some vertex $v \in Q$, then there is a positive constant $c' < \frac{1-\alpha}{2}$, such that whp there are at least $n^{c'}$ vertices in Q that have not chosen l_0.*

We can strengthen the above results by using the following simple observation: For a set of vertices V_2 and $k \geq 2$, let $S^{(k)}_{V_2} \subseteq \mathcal{M}$ denote the set of labels that have been chosen by at least k of the vertices in V_2. Then the probability that every vertex of a set of vertices V_1 is connected to every vertex in V_2 is

$$p(V_1, V_2) \leq \left(|S^{(2)}_{V_2}|p + (1-p)^{|S^{(2)}_{V_2}|} \prod_{v \in V_2} \left(1 - (1-p)^{|S_v - S^{(2)}_{V_2}|}\right)\right)^y \tag{1}$$

$$\leq \left(|S^{(2)}_{V_2}|p + \prod_{v \in V_2} \left(1 - (1-p)^{|S_v|}\right)\right)^y \tag{2}$$

Indeed, the first of the above inequalities corresponds to the probability that each vertex in V_2 either choses one of the labels shared by at least two vertices in V_2, or it is connected to all vertices in V_2 by using labels chosen by exactly one vertex in V_2.

The proof of the following Lemma can be found in the full version of the paper [18].

Lemma 4. *Let $G_{n,m,p}$ be a random instance of the random intersection graphs model with $m = n^\alpha, 0 < \alpha < 1$, $p = \Omega\left(\sqrt{\frac{1}{nm}}\right)$ and $mp^2 = O(1)$. Let also $x = \frac{1}{p^\epsilon}$, for some positive constant $\epsilon < 1$ that can be as small as possible. Then, for any $y \geq np^{1+c}$, where $0 < c < \frac{1-\alpha}{1+\alpha}$ is a constant, we have $\Pr(A_{y,x}) = o(1)$.*

Lemma 4 has the following interpretation (for the proof see the full version of the paper [18]):

Corollary 3. *Let $G_{n,m,p}$ be a random instance of the random intersection graphs model with $m = n^\alpha, 0 < \alpha < 1$, $p = \Omega\left(\sqrt{\frac{1}{nm}}\right)$ and $mp^2 = O(1)$. Let also Q be a clique in $G_{n,m,p}$ that is not formed by a single label and also $|Q| \sim np$. Then whp, for any label $l_0 \in \mathcal{M}$, we have that $|Q \cap L_{l_0}| \leq np^{1+c}$, where $0 < c < \frac{1-\alpha}{1+\alpha}$ is a constant.*

In particular, if Q is not formed by a single label, then whp it is formed by at least $\frac{1}{p^c}$ distinct labels.

Before presenting our main theorem, we provide the following useful lemma, which states that if a large clique is not formed by a single label, then it must contain a quite large clique Q' whose edges are formed by distinct labels. Its proof can be found in the full version of the paper [18].

Lemma 5. *Let $G_{n,m,p}$ be a random instance of the random intersection graphs model with $m = n^\alpha, 0 < \alpha < 1$, $p = \Omega\left(\sqrt{\frac{1}{nm}}\right)$ and $mp^2 = O(1)$. Let also Q be any clique in $G_{n,m,p}$ that is not formed by a single label and also $|Q| \sim np$. Then whp, Q contains a clique Q' whose edges are formed by distinct labels and whose size is at least $p^{-\frac{c}{2}}$, for any positive constant $c < \frac{1-\alpha}{1+\alpha}$.*

We now present our main theorem. Its proof can be found in the full version of the paper [18].

Theorem 3 (Single Label Clique Theorem). *Let $G_{n,m,p}$ be a random instance of the random intersection graphs model with $m = n^\alpha, 0 < \alpha < 1$ and $mp^2 = O(1)$. Then whp, any clique Q of size $|Q| \sim np$ in $G_{n,m,p}$ is formed by a single label. In particular, the maximum clique is formed by a single label.*

Notice that, by Theorem 3, the maximum clique in $G_{n,m,p}$ with $m = n^\alpha, 0 < \alpha < 1$ and $mp^2 = O(1)$ must be one of the sets $L_l, l \in \mathcal{M}$. Therefore, the clique number of $G_{n,m,p}$ can be bounded using the first part of Lemma 2. In particular

Corollary 4. *Let $G_{n,m,p}$ be a random instance of the random intersection graphs model with $m = n^\alpha, 0 < \alpha < 1$, $p = \Omega\left(\sqrt{\frac{1}{nm}}\right)$ and $mp^2 = O(1)$. Then, whp the maximum clique Q of $G_{n,m,p}$ satisfies $|Q| \sim np$.*

5 Label Reconstruction

One of the implications of our main Theorem 3 is that whp we can find the maximum clique in $G_{n,m,p}$ with $m = n^\alpha, 0 < \alpha < 1$ and $mp^2 = O(1)$ in polynomial time, just by looking at the associated bipartite graph $B_{n,m,p}$. More specifically, if we denote by L_i the set of neighbors of label $i \in \mathcal{M}$ in $B_{n,m,p}$, then whp the maximum clique in $G_{n,m,p}$ is any label l such that $L_l = \max_i |L_i|$. Furthermore, given $B_{n,m,p}$ this maximum can be determined in $O(nm)$ time. Therefore, the randomness of the model works in our favor for this case. Indeed, since any graph can be written as an intersection graph with at most $\binom{n}{2}$ labels, the problem of finding a maximum clique in a graph, given its label representation remains NP-complete. Furthermore, it remains hard even when the intersection number is $n^\alpha, 0 < \alpha < 1$ unless the exponential time hypothesis fails (see e.g. [5]).

This leads to the following natural question: Could one infer any information about the structure of the associated bipartite graph when provided with $G_{n,m,p}$ (i.e. the vertices and the edges of the graph)? Notice here that a graph $G_{n,m,p}$ can correspond to more than one associated bipartite graphs. However, we show here that the problem of finding the associated bipartite graph given $G_{n,m,p}$ and the actual values of m, n and p is *solvable* whp when the number of labels is less than the number of vertices; namely, the maximum likelihood estimation method will provide a unique solution (up to permutations of the labels). More specifically, if $\mathcal{B}_{n,m,p}$ is the set of non-isomorphic associated bipartite graphs that give rise to $G_{n,m,p}$, then there is some $B^* \in \mathcal{B}_{n,m,p}$ such that $\Pr(B^*|G_{n,m,p}) > \Pr(B|G_{n,m,p})$, for all $\mathcal{B}_{n,m,p} \ni B \neq B^*$. The proof of the following Theorem can be found in the full version of the paper [18].

Theorem 4. *Let $G_{n,m,p}$ be a random instance of the random intersection graphs model with $m = n^\alpha, 0 < \alpha < 1$, $p = \Omega\left(\sqrt{\frac{1}{nm}}\right)$ and $mp^2 = O(1)$. Then, whp the bipartite graph $B_{n,m,p}$ associated to $G_{n,m,p}$ is uniquely determined, up to permutations of the labels.*

Notice that the uniqueness of the bipartite graph can also be proved in the case where $p = o\left(\sqrt{\frac{1}{nm}}\right)$. Indeed, in this case $G_{n,m,p}$ almost surely has no cycle of size $k \geq 3$ whose edges are formed by k distinct labels (see also the beginning of Section 4). Therefore, every clique of size at least 3 is formed by a single label and so the proof of Theorem 4 applies in this (sparser) case also.

6 Conclusions

In this work, we studied the maximum clique problem by relating it to the intersection number of the input graph. In particular, we first proved that if the intersection number of the graph G is sufficiently small, then a simple algorithm can find a maximum clique in G in polynomial time. We then considered random instances of the random intersection graphs model as input graphs. In particular, by proving the Singe Label Clique Theorem, we provided new, more general and asymptotically tight bounds for the clique number of $G_{n,m,p}$ when $m = n^{\alpha}, \alpha < 1$. We also claim that our proof carries over for $\alpha < 2$, provided there is a lower bound on p (in particular, we claim that our analysis can be applied also for $mp^2 = \Theta(1)$). One of the consequences of our theorem is that we can use the label representation of $G_{n,m,p}$ to find a maximum clique in polynomial time whp. This raised the question of whether we could reconstruct the label choices of the vertices in $G_{n,m,p}$ given only the graph structure. We proved here that the label reconstruction problem is solvable whp when the number of labels is less than the number of vertices. Finding efficient algorithms for constructing such a label representation is left as an open problem for future research. In view of the equivalence results between random intersection graphs and Erdős-Rényi random graphs, we expect that our work will shed light also in the problem of finding maximum cliques for input graphs generated by the latter model.

References

1. Alon, N., Kahale, N.: Approximating the independence number via the θ-function. Math. Programming 80, 253–264 (1998)
2. Alon, N., Krivelevich, M., Sudakov, B.: Finding a large hidden clique in a random graph. Random Structures and Algorithms 13, 457–466 (1998)
3. Ball, F., Sirl, D., Trapman, P.: Epidemics on random intersection graphs. arXiv:1011.4242v1 (math.PR)
4. Behrisch, M., Taraz, A., Ueckerdt, M.: Coloring random intersection graphs and complex networks. SIAM J. Discrete Math. 23, 288–299 (2008)
5. Chen, J., Huang, X., Kanj Iyad, A., Xia, G.: Strong computational lower bounds via parameterized complexity. J. Comput. Syst. Sci. 72(8), 1346–1367 (2006)
6. Coja-Oghlan, A.: A spectral heuristic for bisecting random graphs. Random Structures and Algorithms 29, 351–398 (2006)
7. Coja-Oghlan, A., Lanka, A.: Finding planted partitions in random graphs with general degree distributions. SIAM Journal on Discrete Mathematics 23, 1682–1714 (2009)

8. Feige, U.: Approximating maximum clique by removing subgraphs. SIAM Journal on Discrete Mathematics 18(2), 219–225 (2004)
9. Fill, J.A., Sheinerman, E.R., Singer-Cohen, K.B.: Random intersection graphs when $m = \omega(n)$: an equivalence theorem relating the evolution of the $G(n, m, p)$ and $G(n, p)$ models. Random Struct. Algorithms 16(2), 156–176 (2000)
10. Grimmett, G.R., McDiarmid, C.J.H.: On coloring random graphs. Math. Proc. Cambridge Philos. Soc. 77, 313–324 (1975)
11. Håstad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Mathematica 182, 105–142 (1999)
12. Jerrum, M.: Large cliques elude the metropolis process. Random Structures and Algorithms 3, 347–359 (1992)
13. Juels, A., Peinado, M.: Hiding cliques for cryptographic security. In: Proc. of the Ninth Annual ACM-SIAM SODA, pp. 678–684. ACM Press (1998)
14. Karoński, M., Scheinerman, E.R., Singer-Cohen, K.B.: On random intersection graphs: the subgraph problem. Combinatorics, Probability and Computing Journal 8, 131–159 (1999)
15. Karp, R.M.: Probabilistic analysis of some combinatorial search problems. In: Traub, J.F. (ed.) Algorithms and Complexity: New Directions and Recent Results. Academic Press, New York (1976)
16. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)
17. Nikoletseas, S.E., Raptopoulos, C.L., Spirakis, P.G.: Communication and security in random intersection graphs models. In: WOWMOM, pp. 1–6 (2011)
18. Nikoletseas, S.E., Raptopoulos, C.L., Spirakis, P.G.: Maximum Cliques in Graphs with Small Intersection Number and Random Intersection Graphs. arXiv:1204.4054v1 (cs.DM)
19. Nikoletseas, S.E., Raptopoulos, C.L., Spirakis, P.G.: On the independence number and Hamiltonicity of uniform random intersection graphs. Theor. Comput. Sci. 412(48), 6750–6760 (2011)
20. Robson, J.M.: Finding a maximum independent set in time $O(2^{n/4})$ (2001)
21. Rybarczyk, K.: Equivalence of a random intersection graph and $G(n, p)$. Random Structures and Algorithms 38(1-2), 205–234 (2011)
22. Singer-Cohen, K.B.: Random intersection graphs. PhD thesis, John Hopkins University (1995)

A Finite Basis for 'Almost Future' Temporal Logic over the Reals

Dorit Pardo (Ordentlich) and Alexander Rabinovich

The Blavatnik School of Computer Science, Tel Aviv University, Israel

Abstract. Kamp's theorem established the expressive completeness of the temporal modalities Until and Since for the First-Order Monadic Logic of Order (FOMLO) over Real and Natural time flows. Over Natural time, a single future modality (Until) is sufficient to express all future FOMLO formulas. These are formulas whose truth value at any moment is determined by what happens from that moment on. Yet this fails to extend to Real time domains: Here no finite basis of future modalities can express all future FOMLO formulas. In this paper we show that finiteness can be recovered if we slightly soften the requirement that future formulas must be totally past-independent: We allow formulas to depend just on the very very near-past, and maintain the requirement that they be independent of the rest - actually - of most of the past. We call them 'almost future' formulas, and show that there is a finite basis of almost future modalities which is expressively complete over the Reals for the almost future fragment of FOMLO.

1 Introduction

Temporal Logic (TL) introduced to Computer Science by Pnueli in [Pnu77] is a convenient framework for reasoning about "reactive" systems. This made temporal logics a popular subject in the Computer Science community, enjoying extensive research in the past 30 years. In TL we describe basic system properties by *atomic propositions* that hold at some points in time, but not at others. More complex properties are expressed by formulas built from the atoms using Boolean connectives and *Modalities* (temporal connectives): A k-place modality M transforms statements $\varphi_1 \dots \varphi_k$ possibly on 'past' or 'future' points to a statement $M(\varphi_1 \dots \varphi_k)$ on the 'present' point t_0. The rule to determine the truth of a statement $M(\varphi_1 \dots \varphi_k)$ at t_0 is called a *Truth Table*. The choice of particular modalities with their truth tables yields different temporal logics. A temporal logic with modalities $M_1, \dots, M_k$ is denoted by $TL(M_1, \dots, M_k)$.

The simplest example is the one place modality $\mathsf{F}X$ saying: "X holds some time in the future". Its truth table is formalized by $\varphi_{\mathsf{F}}(t_0, X) \equiv (\exists t > t_0)X(t)$. This is a formula of the First-Order Monadic Logic of Order ($FOMLO$) - a fundamental formalism in Mathematical Logic where formulas are built using atomic propositions $P(t)$, atomic relations between elements $t_1 = t_2$, $t_1 < t_2$, Boolean

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 740–751, 2012.

connectives and first-order quantifiers $\exists t$ and $\forall t$. Most modalities used in the literature are defined by such *FOMLO* truth tables, and as a result every temporal formula translates directly into an equivalent *FOMLO* formula. Thus, the different temporal logics may be considered a convenient way to use fragments of *FOMLO*. *FOMLO* can also serve as a yardstick by which to check the strength of temporal logics: A temporal logic is *expressively complete* for a fragment L of *FOMLO* if every formula of L with a single free variable t_0 is equivalent to a temporal formula.

Actually, the notion of expressive completeness is with respect to the type of the underlying model since the question whether two formulas are equivalent depends on the domain over which they are evaluated. Any (partially) ordered set with monadic predicates is a model for *TL* and *FOMLO*, but the main, *canonical*, linear time intended models are the Naturals $\langle \mathbb{N}, < \rangle$ for discrete time and the Reals $\langle \mathbb{R}, < \rangle$ for continuous time.

A major result concerning *TL* is Kamp's theorem [Kam68, GHR94], which states that the pair of modalities "X *until* Y" and "X *since* Y" is expressively complete for *FOMLO* over the above two linear time canonical models.

Many temporal formalisms studied in computer science concern only future formulas - whose truth value at any moment is determined by what happens from that moment on. For example the formula X *until* Y says that X will hold from now (at least) until a point in the future when Y will hold. The truth value of this formula at a point t_0 does not depend on the question whether $X(t)$ or $Y(t)$ hold at earlier points $t < t_0$.

Over the discrete model $\langle \mathbb{N}, < \rangle$ Kamp's theorem holds also for *future formulas* of *FOMLO*: The future fragment of *FOMLO* has the same expressive power as $TL(\mathsf{Until})$ [GPSS80, GHR94]. The situation is radically different for the continuous time model $\langle \mathbb{R}, < \rangle$. In [HR03] it was shown that $TL(\mathsf{Until})$ is not expressively complete for the future fragment of *FOMLO* and there is no easy way to remedy it. In fact it was shown in [HR03] that there is no temporal logic with a finite set of future modalities which is expressively equivalent to the future fragment of *FOMLO* over the Reals.

The proof there goes (roughly) as follows: Define a sequence of future formulas $\phi_i(z)$ such that given any set B of modalities definable in the future fragment of *FOMLO* by formulas of quantifier depth at most n, the formula $\phi_{n+1}(z)$ is not expressible in $TL(B)$.

The interesting point is that these formulas are all expressible in a temporal language based on the future modality Until plus the modality K^- of [GHR94]. The formula $\mathsf{K}^-(P)$ holds at a time point t_0 if given any 'earlier' t, no matter how close, we can always come up with a t' in between ($t < t' < t_0$) where P holds. This is of course not a future modality - the formula $\mathsf{K}^-(P)$ is past-dependent. And it turns out that not only the above mentioned sequence of future formulas - but all future formulas - can be expressed (over Real time) in $TL(\mathsf{Until}, \mathsf{K}^-)$. This is a consequence of Gabbay's separation theorem [GHR94].

This future-past mixture of Until and K^- is somewhat better than the standard Until - Since basis in the following sense: Although K^- is (like Since) a past modality, it does not depend on much of the past: The formula $\mathsf{K}^-(P)$ depends just on an arbitrarily short 'near past', and is actually independent of most of the past. In this sense we may say that it is an "almost" future formula (see Section 3.1 for precise definitions).

In [HR03] it was conjectured that $TL(\mathsf{Until}, \mathsf{K}^-)$ is expressively complete for almost future formulas of *FOMLO*. Our main result here confirms this conjecture with respect to the Real time domain $(\mathbb{R}, <)$. In the full paper we extend this result to Dedekind complete time flows.

The rest of the paper is organized as follows: In Section 2 we recall the definitions of the monadic logic, the temporal logics and Kamp's theorem. In Section 3.1 we define "almost futureness", then most of the 'machinery' needed for the proof is in Sections 3.2 and 3.3, with the heart of the proof in Lemma 3.13. Section 3.4 then just puts it all together to complete the proof. Finally, Section 4 states further results and comments.

2 Preliminaries

We start with the basic definitions of First-Order Monadic Logic of Order (***FOMLO***) and Temporal Logic (***TL***), and some well known results concerning their expressive power. Fix a ***signature*** (finite or infinite) $\mathcal{S}$ of ***atoms***. We use $P, Q, R, S \ldots$ to denote members of $\mathcal{S}$. Syntax and semantics of both logics are defined below with respect to such a fixed signature.

2.1 First-Order Monadic Logic of Order

Syntax: In the context of *FOMLO*, the atoms of $\mathcal{S}$ are referred to (and used) as ***unary predicate symbols***. Formulas are built using these symbols, plus two binary relation symbols, $<$ and $=$, and a finite set of ***first-order variables*** (denoted by $x, y, z, \ldots$). Formulas are defined by the grammar:

$$atomic ::= \quad x < y \ \mid \ x = y \ \mid \ P(x) \qquad\qquad (\text{where} \ \ P \in \mathcal{S})$$

$$\varphi ::= \quad atomic \ \mid \ \neg\varphi_1 \ \mid \ \varphi_1 \vee \varphi_2 \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \exists x \varphi_1 \ \mid \ \forall x \varphi_1$$

The notation $\varphi(x_1, \ldots, x_n)$ implies that φ is a formula where the x_i's are the only variables occurring free; writing $\varphi(x_1, \ldots, x_n, P_1, \ldots, P_k)$ additionally implies that the P_i's are the only predicate symbols that occur in φ. We will also use the standard abbreviated notation for ***bounded quantifiers***, e.g.: $(\exists x)_{>z}(\ldots)$ denotes $\exists x((x > z) \wedge (\ldots))$, $(\forall x)^{\leq z}(\ldots)$ denotes $\forall x((x \leq z) \rightarrow (\ldots))$, $(\forall x)^{<u}_{>l}(\ldots)$ denotes $\forall x((l < x < u) \rightarrow (\ldots))$, etc.

Semantics: Formulas are interpreted over *structures.* A ***structure*** over $\mathcal{S}$ is a triplet $\mathcal{M} = (\mathcal{T}, <, \mathcal{I})$ where $\mathcal{T}$ is a set - the ***domain*** of the structure, $<$ is an irreflexive partial order relation on $\mathcal{T}$, and $\mathcal{I} : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{T})$ is the ***interpretation*** of the structure (where $\mathcal{P}$ is the powerset notation). We use the standard notation $\mathcal{M}, t_1, t_2, \ldots t_n \models \varphi(x_1, x_2, \ldots x_n)$. The semantics is defined in the standard way. Notice that for ***formulas with a single free first-order variable***, this reduces to:

$$\mathcal{M}, t \models \varphi(x).$$

2.2 Propositional Temporal Logics

Syntax: In the context of *TL*, the atoms of $\mathcal{S}$ are used as ***atomic propositions*** (also called ***propositional atoms***). Formulas are built using these atoms, and a set (finite or infinite) B of ***modality names***, where a non-negative integer ***arity*** denoted by $|\mathsf{M}|$ is associated with each $\mathsf{M} \in B$. The syntax of *TL* with the ***basis*** B over the signature $\mathcal{S}$, denoted by ***TL(B)***, is defined by the grammar:

$$F ::= \quad P \;\mid\; \neg F_1 \;\mid\; F_1 \vee F_2 \;\mid\; F_1 \wedge F_2 \;\mid\; \mathsf{M}(F_1, F_2, \ldots, F_n)$$

where $P \in \mathcal{S}$ and $\mathsf{M} \in B$ an n-place modality (that is, with arity $|\mathsf{M}| = n$). As usual ***True*** denotes $P \vee \neg P$ and ***False*** denotes $P \wedge \neg P$.

Semantics: Formulas are interpreted at ***time-points*** (or ***moments***) in structures (elements of the domain). The domain $\mathcal{T}$ of $\mathcal{M} = (\mathcal{T}, <, \mathcal{I})$ is called the ***time domain***, and $(\mathcal{T}, <)$ - the ***time flow*** of the structure. The semantics of each n-place modality $\mathsf{M} \in B$ is defined by a ‘rule’ specifying how the set of moments where $\mathsf{M}(F_1, \ldots, F_n)$ holds (in a given structure) is determined by the n sets of moments where each of the formulas F_i holds. Such a ‘rule’ for M is formally specified by an operator $\mathcal{O}_{\mathsf{M}}$ on time flows, where given a time flow $\mathcal{F} = (\mathcal{T}, <)$, $\mathcal{O}_{\mathsf{M}}(\mathcal{F})$ is yet an operator in $(\mathcal{P}(\mathcal{T}))^n \longrightarrow \mathcal{P}(\mathcal{T})$.

The semantics of $TL(B)$ formulas is then defined inductively: Given a structure $\mathcal{M} = (\mathcal{T}, <, \mathcal{I})$ and a moment $t \in \mathcal{M}$ (read $t \in \mathcal{M}$ as $t \in \mathcal{T}$), define when a formula F ***holds*** in $\mathcal{M}$ at t - notation: $\mathcal{M}, t \models F$ - as follows:

- $\mathcal{M}, t \models P$ iff $t \in \mathcal{I}(P)$, for any propositional atom P.
- $\mathcal{M}, t \models F \vee G$ iff $\mathcal{M}, t \models F$ or $\mathcal{M}, t \models G$; similarly (“pointwise”) for $\wedge$, $\neg$.
- $\mathcal{M}, t \models \mathsf{M}(F_1, \ldots, F_n)$ iff $t \in [\mathcal{O}_{\mathsf{M}}(\mathcal{T}, <)](T_1, \ldots, T_n)$ where $\mathsf{M} \in B$ is an n-place modality, $F_1, \ldots, F_n$ are formulas and $T_i =_{def} \{s \in \mathcal{T} : \mathcal{M}, s \models F_i\}$.

Truth tables: Practically most standard modalities studied in the literature can be specified in *FOMLO*: A *FOMLO* formula $\varphi(x, P_1, \ldots, P_n)$ (with a single free first-order variable x and with n predicate symbols P_i) is called an ***n-place first-order truth table***. Such a truth table φ ***defines*** an n-ary modality M (whose semantics is given by an operator $\mathcal{O}_{\mathsf{M}}$) iff for any time flow $(\mathcal{T}, <)$, for any $T_1, \ldots, T_n \subseteq \mathcal{T}$ and for any structure $\mathcal{M} = (\mathcal{T}, <, \mathcal{I})$ where $\mathcal{I}(P_i) = T_i$:

$$[\mathcal{O}_M(\mathcal{T}, <)](T_1, \ldots, T_n) = \{t \in \mathcal{T} : \mathcal{M}, t \models \varphi(x, P_1, \ldots, P_n)\}$$

Example 2.1. *Below are truth-table definitions for the well known “**Eventually**”, the (binary)* ***strict*-Until** *and* ***strict*-Since** *of [Kam68] and for* K^- *of [GHR94]:*

- $\Diamond$ *(“**Eventually**”) defined by:* $\varphi_{\Diamond}(x, P) =_{def} (\exists x')_{>x} P(x')$
- **Until** *defined by :* $\varphi_{\mathsf{Until}}(x, P, Q) =_{def} (\exists x')_{>x}(Q(x') \wedge (\forall y)^{<x'}_{>x} P(y))$
- **Since** *defined by:* $\varphi_{\mathsf{Since}}(x, P, Q) =_{def} (\exists x')^{<x}(Q(x') \wedge (\forall y)^{<x}_{>x'} P(y))$
- K^- *defined by:* $\varphi_{\mathsf{K}^-}(x, P) =_{def} (\forall x')^{<x}(\exists y)^{<x}_{>x'} P(y)$

We use infix notation for the binary modalities Until and Since: P Until Q denotes $\mathsf{Until}(P,Q)$, meaning "there is some future moment where Q holds, and P holds all along till then". The ***non-strict*** version Until^{ns} requires that P should hold at the "present moment" as well. The formula $\mathsf{K}^{-}(P)$ holds at the "present moment" t_0 iff given any earlier $t < t_0$ - no matter how close - there is a moment t' in between $(t < t' < t_0)$ where the formula P holds.

2.3 Kamp's Theorem

We are interested in the relative expressive power of TL (compared to $FOMLO$) over the class of ***linear structures***. Major results in this area are with respect to the subclass of ***Dedekind complete structures*** - where the order is Dedekind complete, that is, where every non empty subset (of the domain) which has an upper bound has a least upper bound.

Equivalence between temporal and monadic formulas is naturally defined: $F \equiv \varphi(x)$ iff for any $\mathcal{M}$ and $t \in \mathcal{M}$: $\mathcal{M}, t \models F \Leftrightarrow \mathcal{M}, t \models \varphi(x)$. We will occasionally use $\equiv_{\mathcal{L}}$ / $\equiv_{\mathcal{DC}}$ / $\equiv_{\mathcal{C}}$ to distinguish equivalence over linear / Dedekind complete / any class $\mathcal{C}$ of structures.

Definability: A temporal modality is definable in $FOMLO$ iff it has a $FOMLO$ truth table; a temporal formula F is definable in $FOMLO$ over a class $\mathcal{C}$ of structures iff there is a monadic formula $\varphi(z)$ such that $F \equiv_{\mathcal{C}} \varphi(z)$. In this case we say that φ ***defines*** F over $\mathcal{C}$. Similarly, a monadic formula $\varphi(z)$ may be definable in $TL(B)$ over $\mathcal{C}$.

Expressive completeness/ equivalence: A temporal language $TL(B)$ (as well as the basis B) is expressively complete for (a fragment of) $FOMLO$ over a class $\mathcal{C}$ of structures iff all monadic formulas (of that fragment) $\varphi(z)$ are definable over $\mathcal{C}$ in $TL(B)$. Similarly, one may speak of expressive completeness of $FOMLO$ for some temporal language. If we have expressive completeness in both directions between two languages - they are ***expressively equivalent***.

As Until and Since are definable in $FOMLO$, it follows that $FOMLO$ is expressively complete for $TL(\mathsf{Until}, \mathsf{Since})$. The fundamental theorem of Kamp shows that for Dedekind complete structures the opposite direction holds as well:

Theorem 2.2 ([Kam68]). *$TL(\mathsf{Until}, \mathsf{Since})$ is expressively equivalent to FOMLO over Dedekind complete structures.*

This was further generalized by Stavi who introduced two new modalities Until' and Since' and proved that $TL(\mathsf{Until}, \mathsf{Since}, \mathsf{Until}', \mathsf{Since}')$ and $FOMLO$ have the same expressive power over all linear time flows [GPSS80, GHR94].

2.4 In Search of a Finite Basis for Future Formulas

We use standard ***interval*** notations and terminology for subsets of the domain of a structure $\mathcal{M} = (\mathcal{T}, <, \mathcal{I})$, e.g.: $(t, \infty) =_{def} \{t' \in \mathcal{T} | t' > t\}$; similarly we define $(t, t'), [t, t'), (t, \infty), [t, \infty)$, etc., where $t < t'$ are the ***endpoints*** of the interval. The ***sub-structure*** of $\mathcal{M}$ restricted to an interval is defined naturally. In particular: $\mathcal{M}|_{>t_0}$ denotes the sub-structure of $\mathcal{M}$ restricted to (t_0, ∞): Its domain is

(t_0, ∞) and its order relation and interpretation are those of $\mathcal{M}$, restricted to this interval. $\mathcal{M}|_{\geq t_0}$ is defined similarly with respect to $[t_0, \infty)$. If structures $\mathcal{M}, \mathcal{M}'$ have domains $\mathcal{T}, \mathcal{T}'$, and if I is an interval of $\mathcal{M}$, with endpoints $t_1 < t_2$ in $\mathcal{M}$, such that $I \cup \{t_1, t_2\} \subseteq \mathcal{T} \cap \mathcal{T}'$ and the order relations of both structures coincide on $I \cup \{t_1, t_2\}$ - we will say that I is a ***common interval*** of both structures. This is defined similarly for intervals with ∞ or $-\infty$ as either endpoint. Two structures ***coincide*** on a common interval iff the interpretations coincide there. Two structures ***agree*** on a formula at a given time-point (or along a common interval) iff the formula has the same truth value at that point (or along that interval) in both structures.

Definition 2.3 (Future / past formulas and modalities). *A formula (temporal, or monadic with a single free first-order variable) F is (**semantically**):*

*A **future** formula iff whenever two linear structures coincide on a common interval $[t_0, \infty)$ they agree on F at t_0.*
*A **pure future** formula iff whenever two linear structures coincide on a common interval (t_0, ∞) they agree on F at t_0.*
***Past** and **pure past** formulas are defined similarly.*

*A temporal modality is a first-order **future (past) modality** iff it is definable in FOMLO by a future (past) truth table.*

Note that 'future' can be characterized also syntactically: A formula $\varphi(x_0)$ is a future formula iff it is equivalent to a formula with all quantifiers relativized to $[x_0, \infty)$, that is, all quantifiers are of the form $(\forall x)_{\geq x_0}(\dots)$ or $(\exists x)_{\geq x_0}(\dots)$.

Looking at their truth tables, it is easy to verify that Until is a future modality and Since is a past modality. This pair $\{\mathsf{Until}, \mathsf{Since}\}$ forms an expressively complete (finite) basis in the sense of Kamp's theorem. Do we have a finite basis of future modalities which is expressively complete for all future formulas? Here are some answers:

Theorem 2.4 ([GPSS80]). *$TL(\mathsf{Until})$ is expressively equivalent to the future fragment of FOMLO over discrete time flows (Naturals, Integers, finite).*

Theorem 2.5 ([HR03]). *There is no temporal logic with a finite basis of **future** modalities which is expressively equivalent to the future fragment of FOMLO over Real time flows.*

Theorem 2.6 ([GHR94]). *$TL(\mathsf{Until}, \mathsf{K}^-)$ is expressively complete for the future fragment of FOMLO over Dedekind complete time flows.*[1]

Here we don't have expressive equivalence, as not all $TL(\mathsf{Until}, \mathsf{K}^-)$ formulas are future formulas. Theorem 2.6 offers a finite basis $\{\mathsf{Until}, \mathsf{K}^-\}$, but just like Kamp's $\{\mathsf{Until}, \mathsf{Since}\}$ - this is a 'mixed' future-past basis. [HR03] points out that in spite of its 'past' nature, K^- is "almost" a future modality because it depends just on an arbitrarily small portion of the near past, and is independent of most of the past. It is conjectured there that this "almost future basis" 'generates' only such "almost future formulas", and that it generates ***all*** of them. In this paper we show that this conjecture holds over the Real time domain $(\mathbb{R}, <)$.

[1] This follows [GHR94]'s work along the proof of their separation theorem (10.3.20).

3 A Finite Basis for Almost Future Formulas over $\mathbb{R}$

In Section 3.1 below we define almost future formulas. In Section 3.2 we refine a result of [Hod99], then the most technical part of the proof is in Section 3.3, with the heart of the proof in Lemma 3.13. Section 3.4 finally puts it all together to complete the proof.

3.1 Almost Future Formulas

Definition 3.1 (Almost future formulas, modalities, bases). *A formula (monadic, temporal) F is an almost future formula iff whenever two linear structures coincide on a common interval (t, ∞) they agree on F all along (t, ∞). A temporal modality is almost future iff it has an almost future truth table in FOMLO. A basis is almost future iff all its modalities are.*

Clearly, all pure future formulas are in particular future formulas and all future formulas are almost future. Note that we can give an alternative (equivalent) definition for future and pure future formulas in the style of Definition 3.1 as follows (compare with Definition 2.3): A formula F is

- ***Future*** iff whenever two linear structures coincide on a common interval $[t, \infty)$ they agree on F all along $[t, \infty)$.
- ***Pure future*** iff whenever two linear structures coincide on a common interval (t, ∞) they agree on F all along $[t, \infty)$.

In the sequel we will be interested in "Real structures" - these are structures over time domains isomorphic to the Real time flow $(\mathbb{R}, <)$. We denote this class of structures by $\mathcal{R}$. Note that if $\mathcal{M} \in \mathcal{R}$, then for every $t \in \mathcal{M}$, the structure $\mathcal{M}|_{>t}$ is also in $\mathcal{R}$.

Remark 3.2. *The next two facts and the lemma below follow immediately:*

1. *If an almost future formula holds at t_0 in a substructure $\mathcal{M}|_{>t}$ of some $\mathcal{M} \in \mathcal{R}$ where $t < t_0$ - then it holds there in $\mathcal{M}$ as well.*
2. *If an almost future formula holds at t_0 in a structure $\mathcal{M} \in \mathcal{R}$ then it holds at t_0 in all substructures $\mathcal{M}|_{>t}$ where $t < t_0$.*

Lemma 3.3. *If a basis B is almost future then so are all of $TL(B)$ formulas. In particular:* Until, K^- *and all the formulas of* $TL(\mathsf{Until}, \mathsf{K}^-)$ *are almost future.*

Example: Consider the following property: "Any open interval (t, t_0) contains a proper subinterval (t_2, t_1) such that P (an atomic property) holds at the ends t_1 and t_2, but doesn't hold anywhere inside (t_2, t_1)". This is an almost future property expressible in *FOMLO*. In $TL(\mathsf{Until}, \mathsf{K}^-)$ it is expressed by:

$$\mathsf{K}^-(P \wedge (\neg P \ \mathsf{Until}\ P))$$

Our main result states that, with respect to the class $\mathcal{R}$, ***any*** almost future property expressible in *FOMLO* can be translated to $TL(\mathsf{Until}, \mathsf{K}^-)$:

Main Theorem 3.4. *$TL(\mathsf{Until}, \mathsf{K}^-)$ is expressively equivalent to the almost future fragment of FOMLO over the class of Real structures.*

As Until and K^- are definable in *FOMLO*, the expressive completeness of almost future *FOMLO* for $TL(\mathsf{Until}, \mathsf{K}^-)$ over all linear structures (and in particular over Real ones) follows immediately by Lemma 3.3. For the opposite direction we have to show how almost future monadic formulas translate into $TL(\mathsf{Until}, \mathsf{K}^-)$. Most of our effort will now be in finding such a translation.

In the rest of the paper we highlight the core of the proof, omitting less significant technical details. A detailed proof can be found in the full paper.

3.2 Decomposition Formulas

Both expressive completeness proofs of [GPSS80] (for Theorem 2.4 above) and of [Hod99] (for Kamp's Theorem 2.2) go through manipulating monadic formulas to reach an equivalent formula in some standard form that can then be translated to the target temporal language. We follow the same track:

Definition 3.5 ([Hod99] Decomposition formulas). [2] *A FOMLO formula is **basic** over $TL(B)$ (where B is any temporal basis) iff it is a boolean combination of: (1) Atomic FOMLO formulas and (2) FOMLO formulas definable over Dedekind complete structures in $TL(B)$. A formula of the form: $\exists\bar{x}\forall y\chi(\bar{x}, y, z)$ where $\bar{x}$ is a tuple of first-order variables and χ is basic over $TL(B)$ is called a **decomposition formula** over $TL(B)$.*

Theorem 3.6 ([Hod99]). *Every FOMLO formula $\varphi(z)$ is equivalent over Dedekind complete structures to a positive boolean combination of decomposition formulas over $TL(\mathsf{Until}, \mathsf{K}^-)$:*

$$\varphi(z) \equiv_{\mathcal{DC}} \bigvee_i \bigwedge_j \exists\bar{x}\forall y\chi_{ij}(\bar{x}, y, z), \text{ where } \chi_{ij} \text{ are basic over } TL(\mathsf{Until}, \mathsf{K}^-).$$

Targeting at Kamp's theorem, [Hod99] formulates this theorem and the preceding definition with respect to $TL(\mathsf{Until}, \mathsf{Since})$; yet, the proof there actually uses Since in a restricted form: 'X Since $True$', which is equivalent to $\neg\mathsf{K}^-(\neg X)$. Thus, the proof actually holds for $TL(\mathsf{Until}, \mathsf{K}^-)$ as well.

[GPSS80] introduces a specific form of decomposition formulas where the basic $\chi(\bar{x}, y, z)$ is 'split' into $TL(B)$-definable formulas that 'talk' about a sequence of moments (represented by the tuple $\bar{x}$) and about the sequence of intervals 'marked' by these points:

Definition 3.7 ([GPSS80] $\overleftarrow{\exists}\forall$-formulas). *A FOMLO formula is a $\overleftarrow{\exists}\forall$**-formula** over $TL(B)$ iff it is of the form:*

[2] [Hod99]'s definitions are more general; this simplified version is sufficient for us.

$$
\begin{aligned}
\psi(z) := {} & \exists x_n \ldots \exists x_1 \exists x_0 \\
& [(x_n < x_{n-1} < \cdots < x_1 < x_0 = z) && \text{“\textit{Ordering}”} \\
& \wedge \bigwedge_{j=0}^{n} \alpha_j(x_j) && \text{“\textit{All} } \alpha_j \text{ \textit{'s hold at the points} } x_j\text{”} \\
& \wedge \bigwedge_{j=0}^{n-1} [(\forall y)^{<x_j}_{>x_{j+1}} \beta_j(y)] && \text{“\textit{Each} } \beta_j \text{ \textit{holds along} } (x_{j+1}, x_j)\text{”} \\
& \wedge \ (\forall y)^{<x_n} \beta_n(y)] && \text{“}\beta_n \ holds\ everywhere\ 'before'\ x_n\text{”}
\end{aligned}
$$

where α_j, β_j are FOMLO formulas definable over Dedekind complete structures in $TL(B)$.

Notation 3.8. *Having a particular interest in $\overleftarrow{\exists\forall}$-formulas over $TL(\mathsf{Until}, \mathsf{K}^-)$, we shortly call them $\overleftarrow{\exists\forall}$**-formulas**. We use the notation $\psi^n(z)$ to explicitly reflect the length of the quantifier prefix; and we use the abbreviated notation $\psi^n = (\langle\alpha_0, \beta_0\rangle, \ldots, \langle\alpha_n, \beta_n\rangle)$ for a $\overleftarrow{\exists\forall}$-formula as above, with α_i, β_i definable over Dedekind complete structures in $TL(\mathsf{Until}, \mathsf{K}^-)$.*

The following can be derived from Theorem 3.6 by standard logical equivalences:

Proposition 3.9. *Every FOMLO formula $\varphi(z)$ is equivalent over Dedekind complete structures to a finite disjunction of $\overleftarrow{\exists\forall}$-formulas.*

3.3 Formulas That Hold “Regardless of Most of the Past”

A formula F “holds in $\mathcal{M}$ at t_0 regardless of most of the past” if we can truncate the past as close we wish to the left of t_0, and F persistently holds at t_0 in all such truncated structures. As we will not be using here the dual notion of “holding regardless of most of the future” - we will shortly say that F “almost-holds in $\mathcal{M}$ at t_0”. Formally:

Definition 3.10 (‘Almost holds’). *Given $\mathcal{M} \in \mathcal{R}$ and $t_0 \in \mathcal{M}$, and given a formula (monadic, temporal) F: F **almost-holds** in $\mathcal{M}$ at t_0 iff for every $t < t_0$ in $\mathcal{M}$ there is a $t' \in (t, t_0)$ such that $\mathcal{M}|_{>t'}, t_0 \models F$.*

Remark 3.11.

1. *If a formula F is almost future, $\mathcal{M} \in \mathcal{R}$ and $t_0 \in \mathcal{M}$ then: F holds in $\mathcal{M}$ at t_0 iff it almost-holds there.*
2. *In general, it might be the case that a formula F (which is not almost future) almost-holds in some $\mathcal{M}$ at t_0, yet F does not hold in $\mathcal{M}$ at t_0. Example: “P always held in the past” ($(\forall x)^{<z} P(x)$). Similarly, “P once held in the past” ($(\exists x)^{<z} P(x)$) demonstrates the converse situation.*

Lemma 3.12. *If a finite disjunction of FOMLO formulas $\varphi(z) = \bigvee \psi_i(z)$ is almost future, then for any $\mathcal{M} \in \mathcal{R}$ and $t_0 \in \mathcal{M}$:*

$$\mathcal{M}, t_0 \models \varphi(z) \ \textit{iff some } \psi_i(z) \textit{ almost-holds in } \mathcal{M} \textit{ at } t_0 \qquad (1)$$

Proof. Given an almost future $\varphi(z) = \bigvee \psi_i(z)$ and $t_0 \in \mathcal{M} \in \mathcal{R}$ as above:

Proof of $\Leftarrow$***:*** Let $t < t_0$. Assume that some $\psi_i(z)$ almost-holds in $\mathcal{M}$ at t_0, then there is a $t' \in (t, t_0)$ such that $\mathcal{M}|_{>t'}, t_0 \models \psi_i(z)$, hence $\mathcal{M}|_{>t'}, t_0 \models \varphi(z)$, and as φ is almost future - $\mathcal{M}, t_0 \models \varphi(z)$ as well (Remark 3.2 (1)).

Proof of $\Rightarrow$***:*** Assume that $\mathcal{M}, t_0 \models \varphi(z)$, then (by Remark 3.2 (2)) for every $t < t_0$ in $\mathcal{M}$: $\mathcal{M}|_{>t}, t_0 \models \varphi(z)$, hence for every $t < t_0$:

$$\mathcal{M}|_{>t}, t_0 \models \psi_i(z) \text{ for some index } i \tag{2}$$

Now, assume to the contrary that none of the disjuncts ψ_i almost-holds in $\mathcal{M}$ at t_0. Then for each i there is a point - denote it by t_i - such that $t_i < t_0$ and for all $t' \in (t_i, t_0)$: $\psi_i(z)$ does not hold in $\mathcal{M}|_{>t'}$ at t_0. Let $\bar{t}$ denote the largest ('latest') t_i (we started off with a finite disjunction) and let $t \in (\bar{t}, t_0)$. Then for each i: $t_i \leq \bar{t} < t < t_0$, and therefore for each i: $\psi_i(z)$ does not hold in $\mathcal{M}|_{>t}$ at t_0. This contradicts (2) above. Thus, we conclude that (at least) one of the disjuncts ψ_i ***does*** almost-hold in $\mathcal{M}$ at t_0. □

The above lemma motivates us to seek a way to express in $TL(\mathsf{Until}, \mathsf{K}^-)$ the fact that "a formula almost-holds in $\mathcal{M}$ at t_0". The main technical lemma below shows that this is possible for $\overleftarrow{\exists}\forall$-formulas.

Main Lemma 3.13. *For every $\overleftarrow{\exists}\forall$-formula $\psi(z)$ there is a $TL(\mathsf{Until}, \mathsf{K}^-)$ formula F_ψ such that for every structure $\mathcal{M} \in \mathcal{R}$ and every $t_0 \in \mathcal{M}$:*

$$\mathcal{M}, t_0 \models F_\psi \textit{ iff } \psi(z) \textit{ almost-holds in } \mathcal{M} \textit{ at } t_0 \tag{3}$$

Proof. Let $\psi^n(z) = (\langle \alpha_0, \beta_0 \rangle, \ldots, \langle \alpha_n, \beta_n \rangle)$ be a $\overleftarrow{\exists}\forall$-formula (see Notation 3.8), and let A_i, B_i be $TL(\mathsf{Until}, \mathsf{K}^-)$ formulas defining α_i, β_i ($\alpha_i \equiv_{\mathcal{DC}} A_i$; $\beta_i \equiv_{\mathcal{DC}} B_i$). Define $TL(\mathsf{Until}, \mathsf{K}^-)$ formulas $G_0^{\psi^n}, G_1^{\psi^n}, \ldots, G_n^{\psi^n}, G_{n+1}^{\psi^n}$ and F_{ψ^n} as follows:

$$\begin{aligned}
G_0^{\psi^n} &:= A_0 \\
G_{j+1}^{\psi^n} &:= A_{j+1} \wedge (B_j \ \mathsf{Until}\ G_j^{\psi^n}) \ \text{ - for } j = 0, 1, \ldots, n-1 \\
G_{n+1}^{\psi^n} &:= B_n \ \mathsf{Until}\ G_n^{\psi^n} \\
F_{\psi^n} &:= A_0 \wedge \neg\mathsf{K}^-(\neg B_0) \wedge \bigwedge_{j=1}^{n+1} \mathsf{K}^-(G_j^{\psi^n})
\end{aligned}$$

Now let $t_0 \in \mathcal{M}$, and show that F_{ψ^n} satisfies the required property (3). The $\Leftarrow$ direction follows directly from definitions. For the $\Rightarrow$ direction: Assume that $\mathcal{M}, t_0 \models F_{\psi^n}$. Let $t < t_0$. To show that ψ^n(z) almost-holds in $\mathcal{M}$ at t_0 we must find a $t' \in (t, t_0)$ such that $\mathcal{M}|_{>t'}, t_0 \models \psi^n(z)$.

First, as $\mathcal{M}, t_0 \models \neg\mathsf{K}^-(\neg B_0)$ we have an interval $(\tilde{t}, t_0)$ where B_0 holds and $t < \tilde{t} < t_0$. Second, as $\mathcal{M}, t_0 \models \mathsf{K}^-(G_{n+1}^{\psi^n})$, we have a $t' \in (\tilde{t}, t_0)$ where $G_{n+1}^{\psi^n}$ holds, that is: $\mathcal{M}, t' \models (B_n \ \mathsf{Until}\ G_n^{\psi^n})$. We will find points $t_1, \ldots, t_n, t_{n+1}$ in $\mathcal{M}$ such that (i) $t < \tilde{t} < t' = t_{n+1} < t_n < \cdots < t_1 < t_0$ and (ii) for each $0 \leq i \leq n$: B_i holds in $\mathcal{M}$ along (t_{i+1}, t_i) and $\mathcal{M}, t_i \models G_i^{\psi^n}$ (and thus, in particular

$\mathcal{M}, t_i \models A_i$). Then, as A_i, B_i are almost future (Lemma 3.3), the same holds in the substructure $\mathcal{M}|_{>t'}$ as well (Remark 3.2 (2)). And as $A_i \equiv_{\mathcal{DC}} \alpha_i$; $B_i \equiv_{\mathcal{DC}} \beta_i$, we conclude that $\mathcal{M}|_{>t'}, t_0 \models \psi^n(z)$.

It remains to show there are points t_i as above. For t_{n+1} we simply pick t'. Next, we construct t_n: We have $\mathcal{M}, t_{n+1} \models (B_n \mathsf{Until}\ G_n^{\psi^n})$, hence, $G_n^{\psi^n}$ holds at some $t'' > t_{n+1}$ and B_n holds along (t_{n+1}, t''). Now, if $t'' < t_0$ denote: $t_n = t''$. Otherwise, as $\mathcal{M}, t_0 \models \mathsf{K}^-(G_n^{\psi^n})$, there is a $t^* \in (t_{n+1}, t_0)$ where $G_n^{\psi^n}$ holds - and in this case denote: $t_n = t^*$. In any case, we have $t < \tilde{t} < t' = t_{n+1} < t_n < t_0$, B_n holds along (t_{n+1}, t_n) and $\mathcal{M}, t_n \models G_n^{\psi^n}$. Repeat the above arguments (induction, down-counting from t_n to t_1) to construct the rest of the t_i's. Finally, B_0 clearly holds along (t_1, t_0) and $\mathcal{M}, t_0 \models A_0$, so the points t_i indeed satisfy (i) and (ii) as required. □

3.4 Putting It All Together

Lemma 3.13 renders the desired semantics-preserving translation over Real structures for almost future *FOMLO* formulas. Now we are ready to complete the proof of our main result (Theorem 3.4):

Given an almost future $\varphi(z)$ in *FOMLO*, we will construct a $TL(\mathsf{Until}, \mathsf{K}^-)$ formula F_φ such that:

$$\varphi(z) \equiv_{\mathcal{R}} F_\varphi \tag{4}$$

1. Given an almost future $\varphi(z)$, by Proposition 3.9 we have: $\varphi(z) \equiv_{\mathcal{DC}} \bigvee \psi_i(z)$ where ψ_i are $\overleftarrow{\exists\forall}$-formulas.
2. By Lemma 3.13 each disjunct ψ_i has a 'representative' F_{ψ_i} in $TL(\mathsf{Until}, \mathsf{K}^-)$ that satisfies property (3) of the lemma, or - in other words - that asserts that "$\psi_i(z)$ almost-holds in a Real structure $\mathcal{M}$ at t_0". Define:
$$F_\varphi =_{def} \bigvee F_{\psi_i}$$
3. Notice that so far we haven't used the fact that φ is almost future: Steps 1 and 2 above hold for any monadic $\varphi(z)$. Now verify that (4) above indeed holds: Let $t_0 \in \mathcal{M} \in \mathcal{R}$. By Lemma 3.12 (and this is the point where the "almost futureness" of φ is crucial), $\mathcal{M}, t_0 \models \varphi(z)$ iff there is an index i such that $\psi_i(z)$ almost-holds in $\mathcal{M}$ at t_0, in other words - by Lemma 3.13 - iff there is an i such that $\mathcal{M}, t_0 \models F_{\psi_i}$, that is, iff $\mathcal{M}, t_0 \models F_\varphi$.

4 Further Results and Comments

We have shown expressive equivalence of $TL(\mathsf{Until}, \mathsf{K}^-)$ and almost future *FOMLO* over time flows isomorphic to the Reals. The notion of past, future, almost future formulas is defined with respect to the class of all linear structures. One may as well consider similar notions relative to specific classes of structures. For example, a formula is a future formula over $\mathcal{R}$ (the class of Real structures) if any pair of Real structures that coincide on the future of some point t agree on the formula at t. Clearly, every future formula over the class of all linear structures is also a future formula over $\mathcal{R}$. The converse doesn't hold: "There is a first-moment and P holds there" for example, is unsatisfiable over $\mathcal{R}$, and therefore a future formula over $\mathcal{R}$, but this is not a future formula over Natural time domains. We actually proved a stronger result: Every formula which is almost future over $\mathcal{R}$ has a $TL(\mathsf{Until}, \mathsf{K}^-)$-equivalent over $\mathcal{R}$.

It is decidable whether a formula $\varphi(x)$ is almost future over $\mathcal{R}$. Indeed let $\varphi^{\mathrm{rel}}_{>x'}(x)$ be obtained from φ by relativization of all quantifiers to (x', ∞). A formula φ is almost future over $\mathcal{R}$ iff $\forall x\big((\forall x')^{<x}(\varphi(x) \leftrightarrow \varphi^{\mathrm{rel}}_{>x'}(x))\big)$ is valid over $\mathcal{R}$. Since the validity of a *FOMLO* formula over $\mathcal{R}$ is decidable [BG85], we conclude that it is decidable whether a formula is almost future over $\mathcal{R}$.

In the full paper we prove expressive equivalence of $TL(\mathsf{Until}, \mathsf{K}^-)$ and almost future *FOMLO* over all Dedekind complete structures. Lifting the proof from the Reals to Dedekind complete orders requires careful handling of subtleties that don't appear in the Reals. In Dedekind complete structures there are three types of points: A structure may have a least element or not - a "first moment". A non-first moment is a "successor" if it has a "latest earlier moment" and a "left-limit" otherwise. The fact that in $\mathbb{R}$ all points are left-limits simplifies the proof. The translation presented in Section 3.4 works fine for left-limit points in Dedekind complete structures as well, but fails for successors and first moments. These two types of points need different (but simpler) handling. The core of the proof - handling left-limit points - is the same as presented in this paper.

Over linear structures in general, $\{\mathsf{Until}, \mathsf{K}^-\}$ is not expressive enough: It is not a basis for almost future formulas. Stavi generalized Kamp's theorem by enhancing $\{\mathsf{Until}, \mathsf{Since}\}$ to obtain a basis expressively equivalent to *FOMLO* over linear time [GHR94]. Unfortunately, $\{\mathsf{Until}, \mathsf{K}^-\}$ cannot be extended in a similar manner: In the full paper we show that no finite basis of almost future modalities is expressively equivalent to almost future *FOMLO* over linear time.

Acknowledgments. We are very grateful to Yoram Hirshfeld for numerous insightful discussions, and to the anonymous referees for their helpful suggestions.

References

[BG85] Burgess, J.P., Gurevich, Y.: The decision problem for linear temporal logic. Notre Dame J. Formal Logic 26(2), 115–128 (1985)

[GHR94] Gabbay, D., Hodkinson, I., Reynolds, M.: Temporal logic: Mathematical Foundations and Computational Aspects. Oxford University Press (1994)

[GPSS80] Gabbay, D., Pnueli, A., Shelah, S., Stavi, J.: On the Temporal Analysis of Fairness. In: POPL 1980, pp. 163–173 (1980)

[HR03] Hirshfeld, Y., Rabinovich, A.: Future temporal logic needs infinitely many modalities. Information and Computation 187, 196–208 (2003)

[Hod99] Hodkinson, I.: Notes on games in temporal logic. Lecture Notes for LUATCS Meting, Johannesburg (December 1999), http://www.doc.ic.ac.uk/~imh/index.html

[Kam68] Kamp, H.W.: Tense logic and the theory of linear order. Phd thesis, University of California, Los Angeles (1968)

[Pnu77] Pnueli, A.: The temporal logic of programs. In: Proc. IEEE 18th Annu. Symp. on Found. Comput. Sci., pp. 46–57. IEEE, New York (1977)

Constructing Premaximal Ternary Square-Free Words of Any Level

Elena A. Petrova and Arseny M. Shur

Ural Federal University, Ekaterinburg, Russia
captain@akado-ural.ru, arseny.shur@usu.ru

Abstract. We study extendability of ternary square-free words. Namely, we are interested in the square-free words that cannot be infinitely extended preserving square-freeness. We prove that any positive integer is the length of the longest extension of some ternary square-free word and thus solve an open problem by Allouche and Shallit. We also resolve the two-sided version of this problem.

1 Introduction

Repetition-free words and languages are among the most popular objects of study in formal language theory. Such languages are given by avoidance properties: their words do not contain certain "forbidden" repetitions such as powers, Abelian powers, patterns, palindromes, etc. Lots of challenging problems on these languages are still open. One group of such problems concerns the internal structure of repetition-free languages.

Any repetition-free language can be viewed as a poset with respect to prefix, suffix, or factor order. In case of prefix [suffix] order, the diagram of such a poset is a tree. Each node generates a subtree and is a common prefix [respectively, suffix] of its descendants. Little is known about the structure of these trees. For some power-free languages, it is decidable whether a given word generates finite or infinite subtree [4]. For all kth power-free languages, the subtree generated by any word has at least one leaf [3]. For some particular languages, there is more information. For example, for the binary overlap-free language it is decidable in linear time whether the subtrees generated by two given words are isomorphic [9]. The deciding procedure allows one to check finiteness of a given subtree and some other properties of the tree. In [7] it was proved that binary cube-free words generate arbitrarily large finite subtrees in the corresponding tree. The problem of existence of such subtrees for ternary square-free words was posed in [1, Problem 1.10.9]. Note that considering the factor order instead of the prefix or the suffix one, we get a more general acyclic digraph instead of a tree, but still can ask the same questions about the structure of this digraph. The results of [7, 9] apply to the digraphs of factor order as well.

In this paper we exhibit two infinite series of ternary square-free words. The first series provides the solution to the mentioned problem of [1]; the second series solves the two-sided analog of this problem.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 752–763, 2012.

2 Preliminaries

1. Notation, definitions, auxiliary results. We study words (finite and infinite) and formal languages over the main alphabet $\Sigma = \{a, b, c\}$ and over some auxiliary alphabets. We write Σ^* for the set of all words over Σ including the *empty word* λ, and $|W|$ for the length of the word W. If some letter of Σ is denoted by x, then y and z denote the other two letters. The letters of nonempty finite and right-infinite words are numbered from 1; thus, $W = W[1] \cdots W[|W|]$. The *reversal* of W is the word $(\overleftarrow{W}) = W[|W|] \cdots W[1]$.

We use standard definitions of factors, prefixes, and suffixes of a word. For convenience, the factor $W[i] \cdots W[j]$ is written as $W[i..j]$. A positive integer $p \leq |W|$ is a *period* of a word W if $W[1..|W|-p] = W[1+p..|W|]$. The *exponent* $\exp(W)$ of a word W is the ratio between the length and the minimal period of W. Words of exponent 2 are called *squares*. The *local exponent* of a word is the number $\operatorname{lexp}(W) = \sup\{\exp(V) \mid V \text{ is a factor of } W\}$. A word W is β-free [β^+-free] if $\operatorname{lexp}(W) < \beta$ [respectively, $\operatorname{lexp}(W) \leq \beta$]. The 2-free words are called *square-free*. The language of square-free words over Σ is denoted by SF.

Consider the substitutions α_o ("odd") and α_e ("even") over the alphabet Σ:

$$\alpha_o : \ a \to abc, b \to bca, c \to cab, \quad \alpha_e : \ a \to cba, b \to acb, c \to bac \qquad (1)$$

These substitutions are extended to the function $\alpha : \Sigma^* \to \Sigma^*$ in the following way. To get the image of a word W, substitutions (1) are applied to each letter $W[i]$; if i is odd [even], then the odd [resp., the even] substitution is used. From the definition it follows that the word $\alpha^k(a)$ is a prefix of $\alpha^{k+1}(a)$ for any $k \geq 0$. Hence, one can consider the "limit" right-infinite word

$$\mathsf{A} = \alpha^\infty(a) = abc\,acb\,cab\,cba\,cab\,acb\,cab\,cba\,bca \ldots,$$

which was introduced in [2] and is called the *Arshon word.* We also consider the reversal $\overleftarrow{\mathsf{A}}$ of A. The word A is square-free [2] and, moreover, $(7/4)^+$-free [5]. The factors of A are called *Arshon factors.*

Remark 1. Some Arshon factors occur in A in positions of different parity and thus have two different α-images. We write $\alpha_o(W)$ [$\alpha_e(W)$] for the image of the factor W if W begins in A at an odd [resp., even] position. Obviously, the Arshon factor $\alpha_o(W)$ [$\alpha_e(W)$] begins in an odd [resp., even] position. Thus, W has at most two different α^k-images, denoted by $\alpha_o^k(W)$ and $\alpha_e^k(W)$, for any $k > 0$.

The words U and V are *conjugates* if exists two words X and Y such that $U = XY$ and $V = YX$. If we link up the ends of a word U, we will get a cyclic sequence of letters called a *circular word* and denoted by (U). A circular word represents a conjugacy class of ordinary words. By definition, the factors of (U) are ordinary words of length $\leq |U|$ including U and its conjugates. The word $V = U^2$ is called a *minimal square* if all its proper factors are square-free. Clearly, a word is square-free if and only if there are no minimal squares among its factors. The following proposition shows the role of circular words in the study of square-free words.

Proposition 1 ([8]). *The word U^2 is a minimal square if and only if (U) is square-free.*

2. Formulation of the main result. Let $L \subset \Sigma^*$ and $W \in L$. Any word $U \in \Sigma^*$ such that $UW \in L$ is called a *left context* of W in L. The word W is *left maximal* [*left premaximal*] if it has no nonempty left contexts [respectively, finitely many left contexts]. The *level* of the left premaximal word W is the length of its longest left context[1]. Thus, left maximal words are of level 0. The right counterparts of the above notions are defined in a symmetric way. We say that a word is *premaximal* if it is both left and right premaximal. The *level* of a premaximal word W is the pair $(n, k) \in \mathbb{N}_0^2$ such that n and k are the length of the longest left context of W and the length of its longest right context, respectively.

The aim of this paper is to prove the following theorem:

Theorem 1. *In the language* SF*, there exist*
a) left premaximal words of any level $n \in \mathbb{N}_0$;
b) premaximal words of any level $(n, k) \in \mathbb{N}_0^2$.

3. Overview of the main construction. We prove Theorem 1, a by exhibiting a series of left premaximal words, containing words of any level. The series is constructed in two steps:

1) building an auxiliary series $\{W_n\}_0^\infty$ such that each word W_n has a unique left context of any length $\leq n$;
2) completing the word W_n to a left premaximal word $\widehat{W_n}$.

In order to prove Theorem 1, b, we connect the word $\widehat{W_n}$ to the reversal of the word $\widehat{W_k}$ through some "buffer" word.

If a word $W \in \mathsf{SF}$ has a unique left context of length n, say U, and two left contexts of length $n{+}1$, then we say that U is the *fixed* left context of W. Let us explain step 1. We build the series $\{W_n\}_0^\infty$ inductively. The fixed left context U_n of the word W_n has length $\geq n$. We put $W_0 = cabcacba$ and note that this word has the fixed left context ba of length 2. We require that each word W_n, $n > 0$, has the following properties:

1. W_{n-1} is a prefix of W_n;
2. any suffix of $\overleftarrow{\mathsf{A}}$ is a left context of W_n;
3. the fixed left context U_n of W_n equals $\overleftarrow{\mathsf{A}}[k..1]$ for some $k \geq n$;
4. if $|U_n| > n$, then $W_{n+1} = W_n$ (*trivial* iterations).

If the $(n{+}1)$th iteration is nontrivial, then the fixed context U_{n+1} is longer than $U_n = \overleftarrow{\mathsf{A}}[n..1]$. If $U_n[1] = x$, then both words yU_n and zU_n are left contexts of W_n by definition of the fixed context. Hence, we need to extend W_n to the right in a manner that will "prohibit" the appearance of some letter before U_n in the left context. Let $yU_n = \overleftarrow{\mathsf{A}}[n{+}1..1]$. Then we prohibit z on the $(n{+}1)$th iteration. Proposition 1 suggest the following idea. Find a word S_n such that

[1] Or the height of the subtree generated by W in the tree of suffix order of L.

the circular word $(zU_nW_nS_n)$ is square-free and put $W_{n+1} = W_nS_nzU_nW_nS_n$. Then zU_n is not a left context of W_{n+1} while yU_n is likely to be such a context if the word S_n is chosen appropriately. Thus, if we find an appropriate word S_n for any nontrivial iteration, we are done. For trivial iterations, we artificially put $S_n = \lambda$.

In what follows, the prohibited context zU_n is denoted by U'_{n+1}.

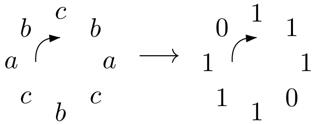
Remark 2. The $(n+1)$th iteration is trivial if U_n starts with *aba*, *abcab*, or *abcbabc* up to renaming the letters. In all other cases, we will be able to provide a nontrivial iteration due to $(7/4)^+$-freeness of the Arshon word. In particular, U'_{n+1} is a square-free word for sure.

3 Codewords and Codewalks

1. Definitions and basic properties. Any ternary square-free circular word U can be encoded by a binary *Pansiot codeword* $\mathsf{cwd}(U)$ of length $|U| - 2$.

$$\mathsf{cwd}(U)[i] = \begin{cases} 0 & \text{if } U[i] = U[i+2], \\ 1 & \text{otherwise;} \end{cases} \quad \text{for example,} \quad \begin{array}{rl} U & = a\,b\,c\,b\,a\,c\,b\,c\ldots\ , \\ \mathsf{cwd}(U) & = 1\,0\,1\,1\,1\,0\quad\ \ldots\ . \end{array}$$

This type of encoding was proposed in [6] for bigger alphabets and studied in [8] for the ternary alphabet. Recall some facts from [8]. First, Pansiot encoding can be naturally extended for circular words. Pansiot codeword for a square-free circular word (U) is the binary circular word $(\mathsf{cwd}(U))$ of length $|U| \geq 3$ obtained as in following example:



Note that one can consider the Pansiot codeword for any ternary word of length ≥ 3 containing no squares of letters. The codewords of square-free (ordinary or circular) words are also called square-free.

Let us consider square-free circular codewords. They do not contain the factors 00 and 1111 encoding the squares of period 2 and 3, respectively. 0's in a codeword correspond to the "jumps" of one letter over another letter in the encoded word. There are six such jumps, represented by the factors *aba*, *bcb*, *cac*, *aca*, *bab*, and *cbc*. We call the first three jumps *right* and the remaining jumps *left*. A right jump in a square-free circular word is always followed by the left jump and vice versa. Thus, the number of 0's in any square-free circular codeword is even. The next jump is obtained from the previous one by

- changing the central letter (e. g., *aba* ↔ *aca*) if the 0's are separated by 1;
- changing the side letters (e. g., *aba* ↔ *cbc*) if the 0's are separated by 11;
- switching the letters (e. g., *aba* ↔ *bab*) if the 0's are separated by 111.

In order to describe square-free codewords, the complete bipartite graph $K_{3,3}$ is used. Left [right] jumps correspond to the bottom [resp., top] part of the graph.

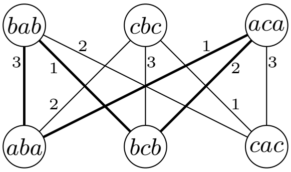


Fig. 1. The graph of jumps in ternary circular words. Bold edges mark the closed codewalk (1213).

The number of 1's between two jumps equals the weight of the edge connecting the corresponding vertices. Each square-free circular codeword of length at least 4 corresponds to a closed walk in the weighted graph shown in Fig. 1.

A walk in the obtained graph is uniquely determined by the initial vertex and the sequence of edge weights. Due to symmetry, this sequence of weights determines whether the walk is closed independently of the initial vertex. Since we are interested in the codewords rather than the encoded words, we consider the walks just as sequences of weights (i. e., words over $\{1, 2, 3\}$), or *codewalks*. Any closed walk is a combination of simple cycles (a closed walk of length two is considered as a simple cycle also). Any simple cycle is defined by a circular word over $\{1, 2, 3\}$. The next lemma is crucial for our considerations.

Lemma 1 ([8]). *A closed walk having (a) no factors* 11*,* 222*,* 223*,* 322*,* 333*, and (b) no factors of the form XYX such that $|Y| = 2$, $|X|$ is even, and (XY) is the label of a closed walk, defines a square-free circular codeword.*

We define a codewalk $\mathsf{cwk}(U)$ for an arbitrary ternary square-free (ordinary) word U as follows. Take the codeword $\mathsf{cwd}(U)$, replace each block of 1's by its length and omit all 0's. For uniqueness, it is necessary to keep the information about the first and the last letters of $\mathsf{cwd}(U)$ (e.g., if $\mathsf{cwd}(U)[1] = 1$, then the first block of 1's in $\mathsf{cwd}(U)$ can be a factor of a longer block). We will underline the first [last] letter of $\mathsf{cwk}(U)$ if $\mathsf{cwd}(U)$ begins [resp., ends] with 1. For example,

$$U = abcacbacaba, \quad \mathsf{cwd}(U) = 110111010, \quad \mathsf{cwk}(U) = \underline{2}31.$$

2. Building the words S_n: closures and buffers. Recall that we look for a word S_n such that the circular word $(zU_nW_nS_n)$ is square-free. This word serves three purposes:

- "close" the corresponding codewalk if it is not closed;
- contain some "buffer" word preventing the appearance of squares after linking up the beginning and the end of the word;
- make the next nontrivial iteration possible (see the following remark).

Remark 3. Since the word $U_nW_{n+1} = U_nW_nS_nzU_nW_nS_n$ is a square without one letter, it has a unique right context of length 1. We add this context *before* adding the buffer word on the next nontrivial iteration.

We describe S_n in terms of codewalks (sometimes, codewords also should be considered). Note that $|\mathsf{cwd}((U))| = |\mathsf{cwd}(U)| + 2$; so, we need to add at least two symbols to the codeword of zU_nW_n. Let $V = \mathsf{cwd}(zU_nW_nS_n)$. Then the codeword $\mathsf{cwd}(W_{n+1})$ is obtained as follows: we take V^2, delete its first $n{+}1$ symbols (which encode U'_{n+1}), and delete the last two symbols (which connect the end of the circular word to its beginning). According to Remark 3, we always begin constructing the codeword of the next buffer word with its first symbol; this symbol encodes the "must" next letter from the previous nontrivial iteration. Since this letter is not z, the added symbol differs from the second last symbol of V (that symbol encodes z in the circular codeword).

Starting from the 7th (3rd nontrivial) iteration, all nontrivial iterations follow the general scheme described in Sect. 4 below. The 1st, 2nd, 5th, and 6th iterations are trivial. The first two nontrivial iterations go as follows:

$$W_2 = W_0 = cabcacba, \quad U'_3W_2 = a\,ba\,cabcacba, \quad \mathsf{cwd}(U'_3W_2) = 010111011.$$

Adding 0111 at the end of this codeword, we obtain the closed codewalk $132\underline{3}$ corresponding to a square-free circular word by Lemma 1. So we double the codeword, delete three first symbols and two last symbols, add the new last symbol and decode the result to get W_3 (for convenience, the codeword of U'_4W_3 required for the next iteration, is shown on the same picture):

$$\begin{array}{lccccccccccccccccccccccccccc}
 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & \\
\mathsf{cwd}(W_3) & & & & & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & & \\
W_3 & & & & & c & a & b & c & a & c & b & a & b & c & a & b & a & c & a & b & c & a & c & b & a & b & c \\
\mathsf{cwd}(U'_4W_3) & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & &
\end{array}$$

The "must" letter after W_3 is b (a is prohibited on this iteration). Now consider the 4th iteration. The codeword $\mathsf{cwd}(U'_4W_3)$ corresponds to a closed codewalk 23231321, but we need to add a symbol 0 to the codeword to encode the "must" letter. Then we add 1110111 to get a closed codewalk $232313213\underline{3}$. By Lemma 1, it encodes a square-free circular word. After decoding, we get $W_4 = W_3bacbca\,bcbaW_3bacbca$.

3. The Arshon word and codewalks. The word $\mathsf{cwk}(\mathsf{A})$ has a specially nice form.

Lemma 2. *Let $f = 132$, $g = 123$, and let B be the right-infinite word generated by the morphism $\beta : \{f,g\}^* \to \{f,g\}^*$ defined by $\beta(f) = fgf$, $\beta(g) = fgg$. Then $\mathsf{cwk}(\mathsf{A}) = \underline{2}\mathsf{B}$ and $\mathsf{cwk}(\overleftarrow{\mathsf{A}}) = \overleftarrow{\mathsf{B}}'1\underline{2}$, where B' is obtained from B by renaming the letters f and g.*

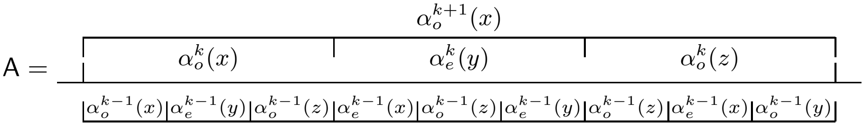
Proof. By Remark 1, for any $k > 0$, the α^k-image of a letter x is determined by the parity of the position of x. We call the words $\alpha_o^k(x)$, $\alpha_e^k(x)$ *α^k-blocks.* Note two obvious properties of the Arshon word and the function α.

(i) α-blocks can be viewed as permutations of Σ: odd α-blocks are permutations of the same parity, while the even α-blocks have the opposite parity.
(ii) x is the first [resp., the last] letter of $\alpha_o(x)$ [resp., of $\alpha_e(x)$].

Assume that $x \in \Sigma$ and $\alpha_o(x) = xyz$. Using (i),(ii), it is easy to check that $\alpha_o^2(x) = xyz\,xzy\,zxy$ and $\alpha_e^2(x) = yxz\,yzx\,zyx$. Note that $\alpha_e^2(x) = \overleftarrow{\alpha_o^2(x)}$. By induction, it is easy to obtain the equality $\alpha_e^k(x) = \overleftarrow{\alpha_o^k(x)}$ for any k.

Now note that in view of (i) all α^2-blocks are equivalent in a sense that they are images of each other under permutations of the alphabet. Thus, any α^2-block has the codeword 1101011 and the codewalk $\underline{2}1\underline{2}$. If we concatenate two α^2-blocks, two symbols should be inserted between their codewords. These two symbols are 01 or 10, because codewords of square-free words avoid 00 and 1111. Then, the codewalk will look like $\underline{2}1321\underline{2}$ or $\underline{2}1231\underline{2}$. Thus, $\mathsf{cwk}(\mathsf{A}) \in \underline{2}(f+g)^*$.

Now consider α^3-blocks. One has $\alpha_o^3(x) = \alpha_o^2(x)\alpha_e^2(y)\alpha_o^2(z)$. Looking at beginnings and ends of these α^2-blocks, we see that the first insertion into the codeword is 10, and the second one is 01. Hence, the codewalk of an odd α^3-block is $\underline{2}1321231\underline{2}$. Since this codewalk is a palindrome and the odd and even α^3-images of a letter are reversals of each other, all α^3-blocks have this codewalk. From the definition of B it follows that if $i \equiv 0 \pmod 3$, then $\mathsf{B}[i] = \mathsf{B}[i/3]$, $\mathsf{B}[i{+}1] = f$, $\mathsf{B}[i{+}2] = g$. So, we have just proved the two last equalities; it remains to prove the first one. To do this, we find the symbols inserted into the codeword at the border of two α^k-blocks, where $k > 2$. Consider an α^{k+1}-block. As above, due to symmetry it is enough to look at a particular odd block:



The first border is in the middle of the factor $\alpha_o^{k-1}(z)\alpha_e^{k-1}(x)$. This factor appears in $\alpha_o^k(z)$, see the picture, and then never appears in an even α^k-block by (i). Thus, the insertion in the codeword at the border of two first α^k-blocks in an α^{k+1}-block is the same as the insertion at the border of two first α^{k-1}-blocks in an α^k-block. The argument for the insertion at the second border of α^k-blocks is the same. Thus, we have confirmed the required condition $\mathsf{B}[i] = \mathsf{B}[i/3]$. The proof of the equality $\mathsf{cwk}(\mathsf{A}) = \underline{2}\mathsf{B}$ is finished. Now the formula for $\mathsf{cwk}(\overleftarrow{\mathsf{A}})$ is straihgtforward from defenitions. □

4. Properties of the word U'_n.

Lemma 3. *Let V be a closed walk which is a prefix of the codewalk $\mathsf{cwk}(U'_n)$, $|V| \geq 6$, and V' is the longest prefix of $\mathsf{cwk}(U'_n)$ with the period $|V|$. Then $|V'| \leq 2|V| - 4$.*

Proof. Recall that the word U'_n is square-free. Since the words $U_n = \overleftarrow{\mathsf{A}}[n..1]$ and U'_n differ only in the first position, the same is true for $\mathsf{cwd}(U'_n)$ and $\mathsf{cwd}(U_n)$. Let us consider the possible prefixes of $\mathsf{cwd}(U_n)$. This codeword cannot have the prefixes 10, 0111, or 110101, because the words 00, 1111, and 010101 encode squares. Also, $\mathsf{cwd}(U_n)$ cannot begin with 01010, 0110110, or 1110111 by Lemma 2. The remaining cases are considered below.

- 01011.... According to Lemma 2, this is the beginning of the block 123 or 132 in the codewalk. Since $\mathsf{cwd}(U'_n) = 11011\ldots$, we have $\mathsf{cwk}(U'_n) = \underline{2}2\ldots$ or $\mathsf{cwk}(U'_n) = \underline{2}32\ldots$. The factors 22, 33, 323, and 232 do not occur inside the codewalk of U_n. Hence, $|V'| \leq |V| + 2 \leq 2|V| - 4$.

- 11011... . By Lemma 2, this is a fragment of 231 or 321 in the codewalk. Changing the first letter, we obtain 121 or 131. These factors do not occur inside $\mathsf{cwk}(U_n)$. Hence, $|V'| \le |V| + 2 \le 2|V| - 4$.
- 0110111... . Here $\mathsf{cwk}(U'_n) = \underline{3}3\ldots$. The factor 33 does not occur inside $\mathsf{cwk}(U_n)$. Hence, $|V'| \le |V| + 1 < 2|V| - 4$.
- 1110110... . Here $\mathsf{cwk}(U'_n) = 22\ldots$. The factor 22 does not occur inside $\mathsf{cwk}(U_n)$. Hence, $|V'| \le |V| + 1 < 2|V| - 4$.
- 011010... . We have $\mathsf{cwk}(U'_n) = \underline{3}1\ldots$.
- 111010... . We have $\mathsf{cwk}(U'_n) = 21\ldots$.

The last two cases are quite similar and we study them simultaneously. Assume that the blocks $f = 132$ and $g = 123$ of $\mathsf{cwk}(U'_n)$ are numbered from left to right starting with 1. Thus, the first block starts in the second position of $\mathsf{cwk}(U'_n)$. Therefore, if $|V'| > |V| + 1$, then $|V|$ is divisible by 3. Since V is a closed walk, $|V|$ is even. So, we conclude that $|V|$ is divisible by 6. It can be checked directly from Fig. 1 that the only closed walks of length 6 that satisfy the partition into blocks f and g are 123123, 132132, and their conjugates. Such roots of length 12 are combinations of 123123, 132132, and conjugates of these combinations. From the description of $\mathsf{cwk}(\mathsf{A})$, an easy calculation shows that the longest prefix of $\mathsf{cwk}(U'_n)$ with the period $|V| = 6$ has length 8 ($2ff1$ or $3gg1$) and the longest prefix of $\mathsf{cwk}(U'_n)$ with the period $|V| = 12$ has length 20 ($3gffggf1$).

Finally, let $|V| = 6k$ for some $k \ge 3$. Note that a block g or f in the codewalk corresponds to the factor of length 9 in the original word. The word V encodes $\frac{6k}{3} \cdot 9 = 18k$ letters, $k \ge 3$. Assume to the contrary that the longest prefix of $\mathsf{cwk}(U'_n)$ with the period $|V|$ has the length at least $2|V|{-}3$. The suffix of V of length 3 encodes at most 10 letters. If U'_n has the prefix of length $36k{-}10$ with the period $18k$, then U_n has a factor of length $36k{-}11$ with this period, namely, the factor $U_n[2..36k{-}10]$. But $(36k - 11)/18k > 7/4$ contradicting to the $(7/4)^+$-freeness of the Arshon word. This contradiction concludes the proof. □

Remark 4. From the proof of Lemma 3 it follows that even stronger statement is true for $\mathsf{cwk}(U_n)$: the inequality $|V'| \le 2|V| - 5$ holds in this case.

Remark 5. Completing the result of Lemma 3, we show that extending U'_n to the left one can avoid short squares.
(1) By Lemma 2, $\mathsf{cwk}(U'_n)$ has no prefixes 11, $\underline{1}1$, 223, 222, $\underline{2}22$, 322, $\underline{3}22$, 333, and $\underline{3}33$. If $\mathsf{cwk}(U'_n) = \underline{2}23\ldots$, we change $\underline{2}$ to 3 (i.e., $\mathsf{cwd}(S_n)$ ends with 1). Thus we avoid the factors listed in Lemma 1,a.
(2) There are three different cycles of length 4 in $K_{3,3}$, they are represented by codewalks 1213, 1232, 1323 and their conjugates. Six of these 12 codewalks can be prefixes of $\mathsf{cwk}(U'_n)$: 1213, 1312, 2123, $\underline{3}132$, $\underline{2}321$, and also $\underline{2}231$ that will be changed to $\underline{3}231$ according to the previous paragraph. The first four codewalks cannot be extended to periodic words due to Lemma 2. $\underline{2}321$ and 3231 can be extended to nasty-looking, in view of Lemma 1,b, codewalks $\underline{2}32123$ and 323132, respectively. In both cases these periodic factors are followed by 1. When $\mathsf{cwk}(U'_n) = \underline{2}32123\ldots$, we change $\underline{2}$ to 3. When $\mathsf{cwk}(U'_n) = 323132\ldots$, we end $\mathsf{cwk}(S_n)$ with 32 and break this period. (Lemma 1 is not a criterion; the codewalk 323132 decodes to a square without one letter.)

4 Constructing Buffer Words and Proving Square-Freeness

The construction of buffer words S_n is based on the following lemma.

Lemma 4. *For any word U_n' there exists a word S_n' such that for any m with the property $\mathsf{cwd}(\mathsf{A})[m{+}1] = 0$ the word $\mathsf{A}[1..m]S_n'$ is the left context of U_n'.*

Proof. We look for a codeword C such that $\mathsf{cwd}(A)[1..m]C\mathsf{cwd}(U_n')$ is a square-free codeword for any m satisfying $\mathsf{cwd}(\mathsf{A})[m{+}1] = 0$. That is, C always starts with 0.. Naturally, we can switch from codewords to codewalks, looking for a codewalk C' such that the codewalk $\mathsf{cwk}(A)[1..m]C'\mathsf{cwk}(U_n')$ decodes to a square-free word.

We know that $\mathsf{cwk}(U_n')$ coincides with $\mathsf{cwk}(U_n)$ except for the first letter, and $U_n = \overleftarrow{\mathsf{A}}[n..1]$. The codewalk $\mathsf{cwk}(U_n')$ has one of the following forms:

1. $21\underline{2}\ldots$
2. $\underline{3}1\underline{2}\ldots$
3. $121\ldots$
4. $131\ldots$
5. $221\ldots$
6. $\underline{3}31\ldots$
7. $\underline{2}23\ldots$
8. $\underline{2}32123\ldots$
9. $\underline{2}32132\ldots$.

The dependence between the forms of $\mathsf{cwk}(U_n')$ at successive nontrivial iterations is shown in Fig. 2. For each vertex, the codewalk C' is written on the outgoing edge. These codewalks are chosen to serve as "Arshon suppressors": their prefixes of length 3 and their suffixes of length 4 (in most cases, 3) do not occur in $\mathsf{cwk}(\mathsf{A})$. Let us show that the codewalk corresponding to the form 1 of $\mathsf{cwk}(U_n')$ satisfies the conditions of the lemma. The proofs for all other cases are similar.

Vertex 1 corresponds to the prefix $\underline{2}1\underline{2}$ and $C' = 3233233$. It is easy to see that the codewalk $\mathsf{cwk}(A)[1..m]C'\mathsf{cwk}(U_n')$ has no factors listed in Lemma 1,a. Furthermore, $\mathsf{cwk}(A)[1..m]C'$ decodes to a square-free word by Lemma 1,b. Indeed, by Remark 4 any suffix XYX of $\mathsf{cwk}(A)[1..m]$ such that XY is a closed walk is of length at most $2|XY| - 5$, and this suffix can be extended by at most two symbols of C', because 323 cannot occur inside $\mathsf{cwk}(A)[1..m]$. Next, $C'\mathsf{cwk}(U_n')$ decodes to a square-free word since the suffix 33 of C' does not occur in $\mathsf{cwk}(U_n')$.

Finally, assume that $\mathsf{cwk}(A)[1..m]C'\mathsf{cwk}(U_n')$ contains a periodic factor XYX that strictly contains C'. Then the factor 3323 of C' belongs to Y, because neither 323 nor 33 can repeat inside the codewalk of the Arshon word. Thus, $|XYX| \le 2|XY| - 4$, and the considered codewalk decodes to a square-free word by Lemma 1,b. After decoding, we obtain a square-free word of the form $\mathsf{A}[1..m]S_n'U_n'$, as required. □

Below we assume that the nth iteration is nontrivial and m is its number among general nontrivial iterations. Using the formula for $\mathsf{cwk}(\mathsf{A})$ (Lemma 2), we take the codewalk $\mathsf{B}[m..2m{-}1]C'\mathsf{cwk}(U_n')$ which decodes to a square-free word by Lemma 4. Consider the codewalk $\mathsf{B}[m..2m{-}1]C'\mathsf{cwk}(U_n'W_{n-1})$. If it is closed, we will prove that it decodes to a square-free circular word. It it is open, we first make it closed, replacing $\mathsf{B}[m..2m{-}1] = \mathsf{cwk}(\mathsf{A})[3m{-}1..6m{-}2]$ by some longer

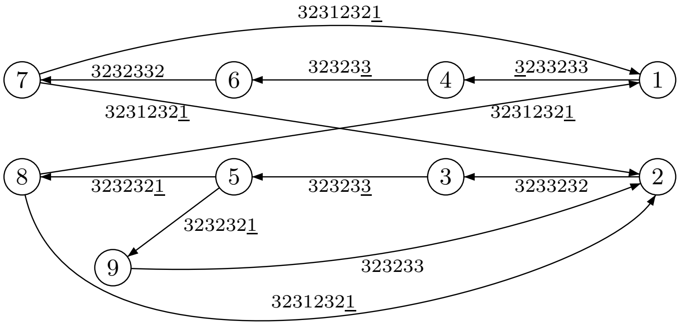


Fig. 2. Nontrivial iterations. Vertices correspond to the forms of $\mathsf{cwk}(U_n')$, edges are marked with the values of C'.

factor $\mathsf{cwk}(\mathsf{A})[3m{-}1..k_m]$. From Fig. 1 it is clear that any two vertices of this graph can be connected with the walk 1, 2, 3, 12, or 13. 1 is the next letter from the Arshon codewalk. Each of the other four walks can be replaced by a factor of $\mathsf{cwk}(\mathsf{A})$ immediately following $\mathsf{B}[m..2m{-}1]$:

2	can be replaced by	123 or 13213 or 1321231
3	can be replaced by	132 or 12312 or 1231321
12	can be used or replaced by	1321
13	can be used or replaced by	1231

(2)

Inserting the appropriate factor after $\mathsf{B}[m..2m{-}1]$ in the considered codewalk, one obtains a closed codewalk $V_m = \mathsf{cwk}(\mathsf{A})[3m{-}1..k_m]C'\mathsf{cwk}(U_n'W_{n-1})$. The prefix $\mathsf{cwk}(\mathsf{A})[3m{-}1..k_m]C'\mathsf{cwk}(U_n')$ of V_m decodes to the square-free word $A_n'S_n'U_n'$, where A_n' is some factor of A. Now consider the border between the factors $\mathsf{cwk}(W_{n-1})$ and $\mathsf{B}[m..2m{-}1]$ in (V_m).

The word W_{n-1} ends with the buffer built at the $(m{-}1)$th general nontrivial iteration from the codewalk ended by some C''. Note that $\mathsf{cwk}(W_{n-1})$ ends with a "distorted" version of C'' according to Remark 3. Namely, the suffix $32\underline{1}$ [resp., 232, $23\underline{3}$, 233 or 332] of C'' was changed to 33 [resp., 231, 231, 232, and 331]. The word $\mathsf{B}[m]$ begins with 1, so if $\mathsf{cwk}(W_{n-1})$ ends with 1, we add 33 to the distorted version of C''. We write $\widetilde{C}_{m-1}$ for the word finally obtained from C''.

Lemma 5. *The following codewalk decodes to a square-free word for any m:*

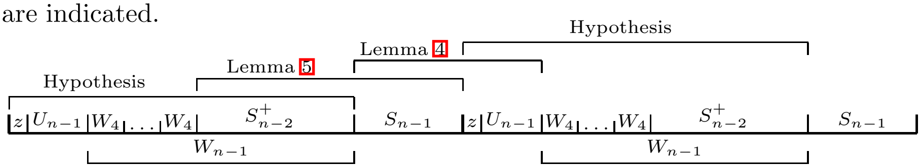
$$D_m = \mathsf{cwk}(\mathsf{A})[2..k_1]\widetilde{C}_1\mathsf{cwk}(\mathsf{A})[5..k_2]\widetilde{C}_2\cdots\mathsf{cwk}(\mathsf{A})[3m{-}1..k_m]\widetilde{C}_m.$$

Proof. Lemma 1 holds for codewalks of ordinary words as well. By construction, D_m has no factors listed in Lemma 1,a. Suppose that D_m has a periodic factor XYX from Lemma 1,b. For $|XY| = 4$, the absence of such factors can be checked directly. Let $|XY| \geq 6$. For short factors, we argue as in the proof of Lemma 4. Recall that such a factor in the Arshon codewalk has the length at most $2|XY| - 5$ and can be extended inside $\widetilde{C}_i$ by at most two letters. If such a factor is contained in $\mathsf{cwk}(\mathsf{A})[3i{-}1..k_i]\widetilde{C}_i\mathsf{cwk}(\mathsf{A})[3i{+}2..k_{i+1}]$ and contains $\widetilde{C}_i$, then at least three letters from $\widetilde{C}_i$ belong to Y. For longer factors, the word Y in XYX is even longer, because the word $\widetilde{C}_i$ can be equal only to some $\widetilde{C}_{i+j}$. But $\widetilde{C}_i = \widetilde{C}_{i+j}$ implies $j \geq 3$ (see Fig. 2), and the fragments of $\mathsf{cwk}(\mathsf{A})$ around $\widetilde{C}_i$ are strictly shorter than their counterparts around $\widetilde{C}_{i+j}$. □

The codewalk $\widetilde{C}_m$ decodes to $S'_n Z_m$ for some word Z_m. We define $S_{n-1} = Z_{m-1}A'_n S'_n$, $S^+_{n-1} = S_6 \cdots S_{n-1}$. Then D_m encodes $S^+_{n-1}Z_m$.

Lemma 6. *The word U_nW_n is square-free.*

Proof. We need to prove two statements: (a) U'_nW_n is a minimal square and (b) U_nW_n does not begin with a square. The considered words are "marked" with the occurrences of W_4 (such occurrences appear only when we double the word W_{i-1} to get W_i at nontrivial iterations). Each W_4 in U_nW_n is preceded by some U'_i, $i \leq n$ and only the first occurrence is preceded by U_n. The latter property accompanied by the direct check for short periods implies (b). In view of Lemma 1, for (a) it suffices to prove that the circular word $(V) = (U'_nW_{n-1}S_{n-1})$ is square-free. We do this by induction on n; the base $n \leq 6$ was established in Sect. 3. Let us prove the inductive step. The following picture demonstrates the structure of V^2. From above, the factors that already proved to be square-free, are indicated.



Let some square $XX = X^{(1)}X^{(2)}$ be a factor of (V). The words S_{n-1} and U'_nW_4 occur only once in (V), so they cannot be factors of X. So, there are only three possible positions for XX: $X^{(2)}$ ends in S_{n-1}; $X^{(2)}$ begins in S_{n-1} and ends in U'_n; $X^{(2)}$ begins in U'_n. In the first case, the part of S^+_{n-1} which belongs to $X^{(2)}$ does not occur to the left, a contradiction. In the second case, the "Arshon suppressing" suffix of S_{n-1} can repeat to the left not later than in S_{n-4}. But then the period of the square is greater than $|S_{n-1}U'_n|$, a contradiction. In the third case, $X^{(2)}$ contains W_4, but there is no room to repeat this factor to the left. This contradiction finishes the proof. □

5 Constructing Premaximal Words

By construction, the word U_n is the fixed left context of W_n. Now we consider the second step, that is, the completion of such "almost uniquely" extendable word W_n to a premaximal word. The main idea is the same as at the first step. In order to obtain a premaximal word of level n, we build the word W_{n+1} in $n{+}1$ iterations and then prohibit the extension of U_nW_{n+1} by the first letter of the word U_{n+1}. We denote the obtained premaximal word of level n by $\overline{W}_n$. Then

$$\overline{W}_n = \underbrace{W_{n+1}\overline{S}_{n+1}}_{}\underbrace{U_{n+1}W_{n+1}\overline{S}_{n+1}}_{}, \qquad (3)$$

where $\overline{S}_{n+1}$ is a buffer word inserted similarly to S_{n+1} in order to close the codewalk and to prevent the appearance of squares at the border. In contrast with the first step, the construction (3) is used only once.

Remark 6. In order to prove Theorem 1,a, it is sufficient to show the existence of left premaximal words of level n for infinitely many different values of n. Indeed, if a word W is left premaximal of level n and $a_1 \cdots a_n W$ is a left maximal word, then the word $a_n W$ is left premaximal of level $n{-}1$.

According to this remark, we construct premaximal words of level n only for the numbers n such that the $(n{+}1)$th iteration is general nontrivial and $\mathsf{cwk}(U'_{n+1}) = 21\underline{2}\ldots$ (vertex 1 in Fig. 2). Then, $U_{n+1} = \underline{3}1\underline{2}\ldots$. Let this iteration be the mth nontrivial general iteration. Then we produce the buffer word from the codewalk $\mathsf{B}[m{+}1..2m{+}1]32332$ (if necessary to close the codewalk, we extend the used factor of the Arshon codewalk according to (2)). The correctness one can check in the same way as for general nontrivial iterations that this insertion leads to a square-free circular word, and then the word $\overline{W}_n$ will have the context U_n of length n but no contexts of bigger length. Thus, Theorem 1,a is proved.

For Theorem 1,b, the rest is easy. We take the words $\overline{W}_n$ and $\overleftarrow{\overline{W}}_k$ and connect them through an Arshon factor of length at least $7(|\overline{W}_n|{+}|\overline{W}_k|)$. The word 32332 in the end of $\mathsf{cwk}(\overline{W}_n)$ changes to 32331 by Remark 3. (No symmetry here: the prefix 23323 of $\mathsf{cwk}(\overleftarrow{\overline{W}}_k)$ does not change.) So, we can take any factor of $\mathsf{cwk}(\mathsf{A})$ which is a product of blocks 321 and 231, and insert between 32331 in the end of $\mathsf{cwk}(\overline{W}_n)$ and 23323 in the beginning of $\mathsf{cwk}(\overleftarrow{\overline{W}}_k)$ to get the required Arshon factor. Thus, the proof of Theorem 1 is finished.

References

1. Allouche, J.P., Shallit, J.: Automatic Sequences: Theory, Applications, Generalizations. Cambridge University Press (2003)
2. Arshon, S.E.: Proof of the existence of asymmetric infinite sequences. Mat. Sbornik 2, 769–779 (1937) (in Russian, with French abstract)
3. Bean, D.A., Ehrenfeucht, A., McNulty, G.: Avoidable patterns in strings of symbols. Pacific J. Math. 85, 261–294 (1979)
4. Currie, J.D.: On the structure and extendibility of k-power free words. European J. Combinatorics 16, 111–124 (1995)
5. Klepinin, A.V., Sukhanov, E.V.: On combinatorial properties of the Arshon sequence. Diskretn. Anal. Issled. Oper. 6(6), 23–40 (1999) (in Russian); English translation in Disc. Appl. Math. 114, 155–169 (2001)
6. Pansiot, J.J.: A propos d'une conjecture de F. Dejean sur les répétitions dans les mots. Discrete Appl. Math. 7, 297–311 (1984)
7. Petrova, E.A., Shur, A.M.: Constructing premaximal binary cube-free words of any level. In: Proc. 8th Internat. Conf. Words 2011 (WORDS 2011). EPTCS, vol. 63, pp. 168–178 (2011)
8. Shur, A.M.: On ternary square-free circular words. Electronic J. Combinatorics 17(# R140) (2010)
9. Shur, A.M.: Deciding context equivalence of binary overlap-free words in linear time. Semigroup Forum 84, 447–471 (2012)

Regularity Problems for Weak Pushdown ω-Automata and Games

Christof Löding and Stefan Repke*

Lehrstuhl für Informatik 7, RWTH Aachen, Germany
{loeding,repke}@automata.rwth-aachen.de

Abstract. We show that the regularity and equivalence problems are decidable for deterministic weak pushdown ω-automata, giving a partial answer to a question raised by Cohen and Gold in 1978. We prove the decidability by a reduction to the corresponding problems for deterministic pushdown automata on finite words. Furthermore, we consider the problem of deciding for pushdown games whether a winning strategy exists that can be implemented by a finite automaton. We show that this problem is already undecidable for games defined by one-counter automata or visibly pushdown automata with a safety condition.

1 Introduction

Finite automaton and pushdown automaton are two of the most fundamental automaton models in computer science. Finite automata have good closure and algorithmic properties. For example, language equivalence and inclusion are decidable (see [9]), and for many subclasses of the regular languages it is decidable whether a given automaton accepts a language inside this subclass (see [17] for some results of this kind). In contrast to that, the situation for pushdown automata is much more difficult. For nondeterministic pushdown automata many problems like language equivalence and inclusion are undecidable (see [9]), and it is also undecidable whether a given nondeterministic pushdown automaton accepts a regular language. The class of languages accepted by deterministic pushdown automata forms a strict subclass of the context-free languages. While inclusion remains undecidable for this subclass, a deep result from [14] shows the decidability of the equivalence problem. Furthermore, the regularity problem for deterministic pushdown automata is also decidable [16, 18].

While automata on finite words are a very useful tool, some applications, in particular verification by model checking (see [2]), require extensions of these models to infinite words. Although the theory of finite automata on infinite words (called ω-automata in the following) usually requires more complex constructions because of the more complex acceptance conditions, many of the good properties of finite automata on finite words are preserved (see [11] for an overview).

* Supported by the DFG Research Training Group 1298 "Algorithmic Synthesis of Reactive and Discrete-Continuous Systems" (AlgoSyn).

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 764–776, 2012.

Pushdown automata on infinite words (pushdown ω-automata) have been studied because of their ability to model executions of non-terminating recursive programs. In [7], efficient algorithms for checking emptiness of Büchi pushdown automata are developed (a Büchi automaton accepts an infinite input word if it visits an accepting state infinitely often during its run). Besides these results, the algorithmic theory of pushdown ω-automata has not been investigated very much. For example, in [6], the decidability of the regularity problem for deterministic pushdown ω-automata has been posed as an open question and to our knowledge no answer to this question is known. Furthermore, it is unknown whether the equivalence of deterministic pushdown ω-automata is decidable.

Our first contribution addresses these questions. We prove the decidability of the two problems (regularity and equivalence) for the subclass of deterministic pushdown ω-automata with weak acceptance condition. Intuitively, an automaton with weak acceptance condition only allows a bounded number of alternations between accepting and rejecting states. This class of automata is capable of expressing boolean combinations of reachability and safety conditions. Our proof is based on a reduction to the corresponding questions for pushdown automata on finite words in the spirit of the minimization algorithm for deterministic weak ω-automata in [10].

We continue our investigations by considering the regularity problem in the extended setting of pushdown games. A pushdown game is given by a pushdown ω-automaton and a partition of the state space into states for Player 0 and Player 1. In a play, the two players build up an infinite sequence of configurations. The partition of the state space determines which player chooses the next successor configuration. In case this automaton is deterministic, the sequences of configurations and sequences of input letters are in one-to-one correspondence, and Player 0 wins if the infinite input word is accepted by the automaton. In this setting, a strategy for a player is a function that tells the player for a given finite sequence of input letters (corresponding to the previous moves of the play) which input letter to choose next (thereby determining the next configuration). It is a winning strategy if each play in which the player follows the strategy is winning for this player. For pushdown games with winning conditions like Büchi condition, or more generally Muller or parity conditions, it is decidable which of the players has a winning strategy, and it is known that such a strategy can be computed by a pushdown automaton with output function reading the letters of the play and outputting the next letter according to the strategy [19].

The regularity problem for pushdown games asks whether the player who has a winning strategy also has one that can be computed by a finite automaton. For example, the question studied in [13, 12], that asks whether for a given document type definition (DTD) one can decide if it is possible to validate streaming XML documents against this DTD with constant memory (by a finite automaton), can be expressed as a regularity problem for a pushdown game with a safety winning condition.

We show that the regularity problem for pushdown games is already undecidable in very simple cases, namely for one-counter automata (pushdown automata with a single stack symbol), and visibly pushdown automata (in which the type of the stack operation is determined by the input letter [1]), both with safety winning conditions. While this result does not transfer back to the constant memory validation question for DTDs, it shows that the latter problem cannot be solved by this more general approach.

The remainder of the paper is structured as follows. In Section 2, we give basic definitions on automata and games. In Section 3, we show the decidability of the regularity and equivalence problem for deterministic weak pushdown ω-automata by a reduction to automata on finite words, and in Section 4 we present our results on the regularity problem for pushdown games.

2 Preliminaries

The set of non-negative integers is $\mathbb{N} := \{0, 1, \dots\}$. For a set S, we denote its cardinality by $|S|$. Let Σ be an alphabet, i.e., a finite set of symbols, then Σ^* (Σ^ω) is the set of *(ω-)words* over Σ, i.e., finite (countably infinite) sequences of Σ symbols. The subsets of Σ^* (Σ^ω) are called *(ω-)languages.* For a word $w = a_1 \cdots a_n \in \Sigma^*$, we define $|w| = n \in \mathbb{N}$ as its length and $w^{\mathrm{R}} = a_n \cdots a_1 \in \Sigma^*$ as its *reversal.* The empty word ε is the word of length $|\varepsilon| = 0$. We assume the reader to be familiar with regular languages, i.e., the languages specified by regular expressions or equivalently by finite state automata. We are mainly concerned with deterministic pushdown automata in this work.

Definition 1. *A* deterministic pushdown machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ *consists of*

- *a finite state set Q, and initial state $q_0 \in Q$,*
- *a finite input alphabet Σ (we abbreviate $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$),*
- *a finite stack alphabet Γ, and initial stack symbol $\bot \notin \Gamma$ (let $\Gamma_\bot = \Gamma \cup \{\bot\}$),*
- *a partial transition function $\delta : Q \times \Gamma_\bot \times \Sigma_\varepsilon \to Q \times \Gamma_\bot^*$ such that for each $p \in Q$ and $A \in \Gamma_\bot$:*
  - *$\delta(p, A, a)$ is defined for all $a \in \Sigma$ and $\delta(p, A, \varepsilon)$ is undefined, or the other way round.*
  - *For each transition $\delta(p, A, a) = (q, W)$ with $a \in \Sigma_\varepsilon$, the bottom symbol $\bot$ stays at the bottom of the stack and only there, i.e., $W \in \Gamma^* \bot$ if $A = \bot$, and $W \in \Gamma^*$ if $A \neq \bot$.*

The set of *configurations* of $\mathcal{M}$ is $Q\Gamma^*\bot$ where $q_0\bot$ is the *initial* configuration. For a given input (ω-)word $w \in \Sigma^*$ ($w \in \Sigma^\omega$), a finite (infinite) sequence $q_0 W_0, q_1 W_1, \dots$ of configurations with $q_0 W_0 = q_0 \bot$ is a *run* of w on $\mathcal{M}$ if there are $a_i \in \Sigma_\varepsilon$ with $w = a_1 a_2 \cdots$ and $\delta(q_i, A, a_{i+1}) = (q_{i+1}, U)$ is such that $W_i = AV$ and $W_{i+1} = UV$ for some stack suffix $V \in \Gamma_\bot^*$.

If the size $|\Gamma|$ of the stack alphabet is 1, then $\mathcal{M}$ is called a *one counter machine.* If the size $|\Gamma|$ of the stack alphabet is 0, then $\mathcal{M}$ is called a *finite state machine,* and we omit the components related to the stack from the notation of the machine and the transitions.

Automata and Languages. For finite words, we consider the model of a *deterministic pushdown automaton (DPDA)* $\mathcal{A} = (\mathcal{M}, F)$ consisting of a deterministic pushdown machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ and a set of final states $F \subseteq Q$. It *accepts* a word $w \in \Sigma^*$ if w induces a run ending in a final state. These words form the language $L_*(\mathcal{A}) \subseteq \Sigma^*$. For ω-words, we use the model of a *deterministic parity pushdown automaton (ω-DPDA)* $\mathcal{A} = (\mathcal{M}, \tau)$ consisting of a deterministic pushdown machine $\mathcal{M}$ as above and a function $\tau : Q \to \mathbb{N}$ assigning *colors* to the states of $\mathcal{M}$. An infinite word $\alpha \in \Sigma^\omega$ is *accepted* if it induces an infinite run such that the lowest color of the states occurring infinitely often is even. The accepted ω-words form the ω-language $L_\omega(\mathcal{A}) \subseteq \Sigma^\omega$. We call an ω-DPDA *weak* if colors never increase during a run. When restricting the color set of a (weak) ω-DPDA to $\{0, 1\}$ or $\{1, 2\}$, we end up with *Büchi (reachability)* and *coBüchi (safety)* acceptance, respectively.

To ensure that an infinite run reads an infinite word, we require that $\mathcal{A}$ has no infinite sequence of ε-transitions. The presence of such sequences can be tested, and they can be removed in polynomial time by redirecting some of the ε-transitions into sink states. Under this assumption, for a finite word w, we define $\delta_*(w)$ to be the last state q_n of a run $q_0 W_0, \ldots, q_n W_n$ on w such that there is no further ε-transition possible.

All restrictions in the type of the underlying pushdown machine carry over to the automata. Finite state (ω-)automata are denoted by (ω-)DFA. As usual, an (ω-)language is called regular if it can be accepted by a (ω-)DFA.

Games and Strategies. A *pushdown game (PDG)* $\mathcal{G} = (\mathcal{M}, \tau, Q_0)$ consists of an ω-DPDA $(\mathcal{M}, \tau)$ and a set $Q_0 \subseteq Q$. A *play* is an ω-word $\alpha = a_1 a_2 \cdots \in \Sigma^\omega$ successively build up by two players. After the prefix $a_1 \cdots a_i$, the next action $a_{i+1} \in \Sigma$ is chosen by Player 0 if $\delta_*(a_1 \cdots a_i) \in Q_0$, otherwise Player 1 chooses (since $\mathcal{M}$ is total). Player 0 wins the play α iff it is accepted by the ω-DPDA $(\mathcal{M}, \tau)$. This can be considered as the game with the (possibly infinite) configuration graph of $\mathcal{M}$ as arena. A *strategy* is a function $f : \Sigma^* \to \Sigma$ advising to choose action $f(w)$ after a finite play prefix $w \in \Sigma^*$. We call it *winning* for a Player if he wins a play as long as he obeys to f no matter what his opponent does. A game can be *won* by a player if he has a winning strategy. We are especially interested in winning strategies representable by automata. A *pushdown strategy (PDS)* $\mathcal{F} = (\mathcal{M}', \sigma)$ consists of a deterministic pushdown machine $\mathcal{M}'$ and a function $\sigma : Q' \to \Sigma$ advising actions according to the states of $\mathcal{M}'$. It defines the strategy $f(w) = \sigma\big(\delta_*(w)\big)$.

Again the definitions carry over for the restricted classes of finite state and one counter machines. Finite state games and one counter games are denoted by FSG and 1CG. In general, the winning player of a PDG has a winning pushdown strategy [19], and the winner of an FSG has a winning finite state strategy (FSS) [3].

A pushdown game that is useful in several places in this work is the so called classification game associated to an ω-DPDA $\mathcal{A}$ and a set C of colors. The idea in this game is that one player plays an infinite input word and the other player plays an infinite sequence of colors from C. The configurations of the game mimic the run of $\mathcal{A}$ on the input word played. The player who is in charge of the colors wins with an accepting color sequence iff the ω-DPDA accepts the input word played. Since we are interested in weak automata, we consider the weak classification game in which the player in charge of the colors is not allowed to increase the colors during a play.

Definition 2. *Let $C \subseteq \mathbb{N}$ be finite and $\mathcal{A} = (\mathcal{M}, \tau)$ with $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ be an ω-DPDA. We define the* weak classification game $\mathcal{G}_{\mathcal{A},C} = (\mathcal{M}', \tau', Q'_0)$ *with $\mathcal{M}' = (Q', \Sigma', \Gamma', \delta', q'_0, \bot')$ as follows.*[1]

- $Q' = Q \times C \times \{0,1\}$, $q'_0 = \big(q_0, \max(C), 0\big)$, *and* $(q,i,x) \in Q'_0$ *iff* $x = 0$,
- $\Sigma' = \Sigma \uplus C$, $\Gamma' = \Gamma$, $\bot' = \bot$,
- $\delta'\big((q,i,0), A, j\big) = \big((q,j,1), A\big)$, *where* $j \in \{0, \dots, i\}$,
  $\delta'\big((q,i,1), A, \varepsilon\big) = \big((p,i,1), W\big)$, *if* $\delta(q, A, \varepsilon) = (p, W)$, *and otherwise*
  $\delta'\big((q,i,1), A, a\big) = \big((p,i,0), W\big)$, *if* $\delta(q, A, a) = (p, W)$ *for* $a \in \Sigma$,
- $\tau'\big((q,i,x)\big) = \big(\tau(q) + i\big)$

Note that the winning condition is a parity condition in general which is weak iff $\mathcal{A}$ is weak. Since the second component in the vertices representing the color from C can never increase, it remains fixed to some i from some point on in every play. Then the original parity condition of $\mathcal{A}$ is evaluated shifted by this fixed index i. If i is even, then Player 0 wins if the input word is accepted by $\mathcal{A}$. If i is odd, then Player 0 wins if the input word is rejected by $\mathcal{A}$. Thus, the acceptance status of the color sequence played by Player 0 has to correspond to the acceptance status of the played input word. Therefore, if Player 0 has a winning strategy in the game that can be implemented by some kind of machine (pushdown or finite state), then this strategy can directly be transformed into a weak automaton for $L_\omega(\mathcal{A})$ by using the color output of the strategy as colors for the acceptance condition.

A first application of this game is the observation that the computation power of the automaton model (pushdown or finite state) and the expressiveness of the acceptance condition are in some sense orthogonal. This statement can also be derived from the proof of Theorem 24 in [15].

Lemma 3. *Let $\mathcal{A}$ be a weak ω-DPDA such that $L_\omega(\mathcal{A})$ is regular. Then there exists a weak ω-DFA $\mathcal{A}'$ with the same colors as $\mathcal{A}$ such that $L_\omega(\mathcal{A}') = L_\omega(\mathcal{A})$.*

Proof. Let $\mathcal{A}$ and $L = L_\omega(\mathcal{A})$ be as above, and $\mathcal{A}''$ be an ω-DFA with $L_\omega(\mathcal{A}'') = L$. Consider the weak classification game $\mathcal{G}_{\mathcal{A}'',C}$ with $C = \tau(Q)$ being the colors of $\mathcal{A}$. Player 0 can easily win $\mathcal{G}_{\mathcal{A}'',C}$ by a pushdown strategy simulating $\mathcal{A}$, i.e., always playing the colors according to $\mathcal{A}$. Since $\mathcal{A}''$ is a finite automaton, $G_{\mathcal{A}'',C}$ is a finite state game, and Player 0 also has a finite state strategy [3] which can be used as a weak ω-DFA $\mathcal{A}'$ with colors C such that $L_\omega(\mathcal{A}') = L$. □

[1] For readability, we dropped the definition of a positive and negative sink state for cases in which a wrong action is played, and the corresponding player has to lose.

3 Regularity and Equivalence Testing for Weak ω-DPDA

In this section, we show how to decide the regularity and the equivalence problems for weak ω-DPDAs. The equivalence problem asks for two given weak ω-DPDAs whether they accept the same language, and the regularity problem asks for a given weak ω-DPDA whether it accepts a regular ω-language. Both problems are decidable in the case of DPDAs over finite words ([14] for equivalence and [16, 18] for regularity). We show that the problems for weak ω-DPDAs can be reduced to the case of finite words. Our technique is inspired by a normal form of weak ω-DFA which assigns minimal colors to each state [10]. Then certain decision problems for ω-DFA can be reduced to DFA on finite words. The same approach for weak ω-DPDA faces the difficulty that minimal colors also depend on the stack content. We overcome this by considering minimal colors for configurations which yields a regular coloring. This also allows us to normalize weak ω-DPDA such that we can apply known algorithms for finite words.

For the remainder of this section, let $\mathcal{A} = (\mathcal{M}, \tau)$ be a weak ω-DPDA with $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ and greatest color $k = \max\big(\tau(Q)\big)$. W.l.o.g., we assume $k \geq 1$. The first important step is to define a language of finite words from the ω-language of $\mathcal{A}$ by simply setting states with an accepting (even) color as final.

Definition 4. *The* finitary language $L_*(\mathcal{A}) \subseteq \Sigma^*$ *of* $\mathcal{A}$ *is the language of finite words accepted by the DPDA* $(\mathcal{M}, F)$ *with* $F = \big\{q \in Q \,\big|\, \tau(q) \text{ is even}\big\}$.

By $L_*(\mathcal{A}_{qW})$ and $L_\omega(\mathcal{A}_{qW})$, we denote the respective languages recognized by $\mathcal{A}$ starting from configuration qW. The following observation is a direct consequence of the definitions.

Remark 5. If $L_*(\mathcal{A}_{qW}) = L_*(\mathcal{A}_{pV})$, then $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$.

If the inverse would also be true, then we would have established a strong relation between the ω-language of a weak ω-DPDA and its language of finite words. Unfortunately, this is not true in general, as illustrated by the following example.

Example 6. For $n \in \mathbb{N}$, consider the language $L_n \subseteq \Sigma^\omega$ over alphabet $\Sigma = \{a, b, c\}$ defined as follows:

$$L_n = \bigcup_{x \in \{a,b\}} \Big(x\{a,b\}^* x\{a,b\}^n c \Sigma^\omega \Big).$$

Obviously, L_n is a regular ω-language since it is defined by a regular expression. Every ω-DFA recognizing L_n needs a state set of size at least exponential in n because before reading the first c it has to remember the last $n+1$ symbols. A weak ω-DPDA with a state set of size linear in n and a stack alphabet of constant size can recognize L_n by writing the string $w \in \{a, b\}$ that occurs before the first c onto the stack (using state q_0). Afterwards, one can check the reversal w^{R} of w with linearly many states by counting to $n+1$ (using the states $q_1, \ldots, q_{n+1}$), then moving to q_a or q_b depending on the letter on the stack, and comparing this letter with the bottom letter on the stack (moving to $q_\top$ or $q_\bot$ to indicate the result of the comparison). To identify the bottom letter, it is stored as $\$_a$ or $\$_b$ while the other letters are stored as $\#_a$ or $\#_b$ on the stack.

The full definition of the weak ω-DPDA $\mathcal{A}_n = \big((Q, \Sigma, \Gamma, \delta, q_0, \bot), \tau\big)$ is: $Q = \{q_0, \dots, q_{n+1}, q_a, q_b, q_\top, q_\bot\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{\$_a, \$_b, \#_a, \#_b\}$, and transitions as follows (where $A \in \Gamma_\bot$, $x, y \in \{a, b\}$, $z \in \Sigma$, $i \in \{1, \dots, n\}$):

- $\delta(q_0, A, x) = \begin{cases} (q_0, \$_x A) & \text{if } A = \bot, \\ (q_0, \#_x A) & \text{if } A \neq \bot, \end{cases}$
- $\delta(q_0, A, c) = (q_1, A)$, $\delta(q_i, \#_x, z) = (q_{i+1}, \varepsilon)$, $\delta(q_{n+1}, \#_x, z) = (q_x, \varepsilon)$,
- $\delta(q_x, \#_y, z) = (q_x, \varepsilon)$, $\delta(q_x, \$_y, z) = \begin{cases} (q_\top, \varepsilon) & \text{if } x = y, \\ (q_\bot, \varepsilon) & \text{if } x \neq y, \end{cases}$
- $\delta(q_\top, A, z) = (q_\top, A)$, and for all other transitions: $\delta(q, A, z) = (q_\bot, A)$,

and reachability coloring $\tau : Q \to \{0, 1\}$ (where $i \in \{1, \dots, n+1\}$): $\tau(q_\top) = 0$ and $\tau(q_0) = \tau(q_i) = \tau(q_a) = \tau(q_b) = \tau(q_\bot) = 1$.

Then we have $L_\omega(\mathcal{A}_n) = L_n$. To see that the inverse of Remark 5 does not hold, note that in configurations with a state from $\{q_1, \dots, q_{n+1}, q_a, q_b\}$, it is already clear whether the remaining word is accepted or not (this only depends on the stack content). For example from configurations $q_a W \$_a \bot$ and $q_b V \$_b \bot$, all infinite words are accepted but the set of finite words accepted depends on the height of the stack and is thus different if W and V are of different length.

Furthermore, $L_*(\mathcal{A}_n)$ is not regular because a finite word is only accepted if it reaches $q_\top$, which requires as many letters after the first c as before the first c:

$$L_*(\mathcal{A}_n) \;=\; \bigcup_{x \in \{a,b\}} \Big(x\{a, b\}^i x \{a, b\}^n c \Sigma^{1+i+1+n} \Sigma^* \Big).$$

Our next goal is to normalize $\mathcal{A}$ such that the inverse of Remark 5 becomes true. Therefore, we redefine the coloring of configurations (not simply states) based on the sets K_i defined below. The intuition behind the definition is that each configuration should be assigned the lowest color possible.

Definition 7. *We partition the configurations of a* $\mathcal{A}$ *into* classes K_i *for* $i \in \mathbb{N}$*, where* K_i *is the biggest subset of* $(Q\Gamma^*\bot) \setminus (\bigcup_{j<i} K_j)$ *such that:*

a) each run staying in K_i *forever is accepting iff* i *is even, and*
b) each run leaving K_i *leaves it to* $\bigcup_{j<i} K_j$*.*

Example 8. The classes K_i for $\mathcal{A}_n$ from Example 6 are (where $\# = \{\#_a, \#_b\}$):

$$\begin{aligned} K_0 &= (q_\top \Gamma^* \bot) \cup \bigcup_{x \in \{a,b\}} \Big(q_x \#^* \$_x \Gamma^* \bot \Big) \cup \bigcup_{\substack{x \in \{a,b\} \\ i \in \{0,\dots,n\}}} \Big(q_{n+1-i} \#^i \#_x \#^* \$_x \Gamma^* \bot \Big), \\ K_1 &= (Q\Gamma^*\bot) \setminus K_0, \quad \text{for } i \geq 2. \end{aligned}$$

Note that K_0 are exactly the configurations from where every word is accepted. All other configurations belong to K_1, like the ones with the bottom state ($q_\bot \Gamma^* \bot \subseteq K_1$) but also the initial state ($q_0 \Gamma^* \bot \subseteq K_1$). Further, some states occur in both K_0 and K_1, like $q_a \$_a \bot \in K_0$ but $q_a \$_b \bot \in K_1$, or $q_1 \#_a \#^n \$_a \bot \in K_0$ but $q_1 \#_a \#^n \$_b \bot \in K_1$.

Definition 9. *$\mathcal{A}$ is in* normal form *if colors correspond to classes, i.e., $qW \in K_{\tau(q)}$ for all $qW \in Q\Gamma^*\bot$ that are reachable from the initial configuration.*

Example 8 shows that $\mathcal{A}_n$ is not in normal form. However, the classes K_0 and K_1 are regular sets of words. This is true in general and can be used to transform $\mathcal{A}$ into normal form.

Lemma 10. *For $\mathcal{A}$, one can compute in exponential time a weak ω-DPDA $\mathcal{A}'$ in normal form such that $L_\omega(\mathcal{A}') = L_\omega(\mathcal{A})$, and a DPDA $\mathcal{A}''$ such that $L_*(\mathcal{A}'') = L_*(\mathcal{A}')$, where $\mathcal{A}''$ has $\mathcal{O}(|Q|)$ states and $|\Gamma|\cdot 2^{\mathcal{O}(|Q|\cdot k)}$ stack symbols.*

Proof (Sketch). The core of the proof consists in constructing a deterministic finite state machine that reads a reversed configuration and assigns the corresponding class K_i to it. This can be achieved by considering the weak classification game $G_{\mathcal{A},C}$ where C is the set of colors of $\mathcal{A}$. Obviously, Player 0 has a winning strategy (by simply playing the colors from the unique run on the input word played by Player 1). From the definition of the K_i one can deduce that the smallest i such that Player 0 wins from a position $(qW, i, 0)$ is such that $qW \in K_i$. The set of configurations from which Player 0 has a winning strategy is regular and can be accepted by an alternating automaton with a number of states linear in the number of states of the pushdown machine defining $G_{\mathcal{A},C}$ [4, 8]. This alternating automaton can be turned into a DFA of exponential size reading the reversed configurations [5]. This DFA can then be simulated on the configurations along a run of $\mathcal{A}$ by storing its states on the stack in parallel to the actual stack symbols, which yields the normalized ω-DPDA $\mathcal{A}'$. Therefore, the blowup of the stack alphabet is exponential. The states of $\mathcal{A}'$ only contain the additional information on the class K_i the configuration is in. Finally, for the DPDA $\mathcal{A}''$, this information is reduced to whether the class is even or odd, resulting in $2\cdot|Q|$ many states. Each step of the computation can be done in exponential time. □

The example illustrates that the deterministic automaton reading the configurations in reverse has to be of exponential size in general. For weak ω-DPDA in normal form, the inverse of Remark 5 is true.

Lemma 11. *Let $\mathcal{A}$ be in normal form. If $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$, then $L_*(\mathcal{A}_{qW}) = L_*(\mathcal{A}_{pV})$.*

Proof (Sketch). Using basic properties on the sets K_i, a simple argument shows that if $\mathcal{A}$ is in normal form, then two configurations qW and pV with $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$ must have the same color, i.e., $\tau(q) = \tau(p)$. From this, the statement of the lemma easily follows. □

Remark 5 and Lemma 11 can be summarized as follows when considering the special case of initial configurations of two weak ω-DPDA.

Corollary 12. *Let $\mathcal{A}$ and $\mathcal{A}'$ be weak ω-DPDA in normal form. $L_*(\mathcal{A}) = L_*(\mathcal{A}')$ iff $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$.*

The next result is an immediate consequence since the equivalence problem for DPDA is decidable [14].

Corollary 13. *The equivalence problem for weak ω-DPDA is decidable.*

We can also use Corollary 12 to reduce the regularity problem to the case of finite words.

Theorem 14. *Let $\mathcal{A}$ be in normal form. $L_*(\mathcal{A})$ is regular iff $L_\omega(\mathcal{A})$ is regular.*

Proof. For the implication from finite to infinite words, suppose $L_*(\mathcal{A})$ is regular, i.e., $L_*(\mathcal{A}) = L_*(\mathcal{A}')$ for some DFA $\mathcal{A}'$. Viewing $\mathcal{A}'$ as a Büchi automaton (assigning color 1 to rejecting and color 0 to accepting states), we obtain that $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$ because $\mathcal{A}'$ visits a state of color 0 after the same prefixes as $\mathcal{A}$ visits a state with even color. Therefore, $L_\omega(\mathcal{A})$ is regular (this also shows that $\mathcal{A}'$ is a weak ω-DFA for an appropriate coloring).

If $L_\omega(\mathcal{A})$ is recognizable by an ω-DFA, then Lemma 3 shows that there is even a weak such ω-DFA $\mathcal{A}'$. We can assume that $\mathcal{A}'$ is in normal form (using our construction or the results in [10]). Applying Corollary 12, we obtain that $L_*(\mathcal{A}) = L_*(\mathcal{A}')$ is regular. This shows the inverse implication. □

Our main result uses the fact that regularity for DPDA is decidable [16, 18].

Corollary 15. *The regularity problem for weak ω-DPDA is decidable.*

Complexity. Our method to decide regularity is composed of transforming weak ω-DPDA into normal form and, applying the known regularity test for DPDA on its finitary language. Finally, in case of a positive result, the DFA being equivalent to the finitary language can be colored appropriately according to Theorem 14, to yield a weak ω-DFA that recognizes the original ω-language.

Lemma 10 normalizes a given weak ω-DPDA $\mathcal{A}$ in exponential time, and yields a DPDA $\mathcal{A}''$ that recognizes its finitary language, with $\mathcal{O}(|Q|)$ states and $|\Gamma|\cdot 2^{\mathcal{O}(|Q|^2)}$ stack symbols since we assume $k \in \mathcal{O}(|Q|)$. For the regularity test in the second step, we refer to [18] which gives better complexity bounds than [16]. According to Valiant's work, a DPDA that recognizes a regular language with n states, t stack symbols, and words of at most length h in transitions, can be transformed in doubly exponential time into an equivalent DFA with $E^2(n^2 \log n + \log t + \log h)$ states, where $E^i(f) = \exp^i\big(\mathcal{O}(f)\big)$ denotes an exponentiation tower of height i. Both steps in composition run in triply exponential time. In case that the ω-language is regular, it yields a DFA, the number of states of which is bounded by $E^2\big(|Q|^2 \log |Q| + \log |\Gamma| + \log h\big)$, and which can be colored appropriately to yield a weak ω-DFA equivalent to $\mathcal{A}$.

Hence, the regularity test for ω-languages presented here is more expensive in terms of computation time than for languages of finite words. Nevertheless, in case of a regular ω-language, the resulting weak ω-DFA has the exact same bound on the number of states as for finite words. From Example 8, we see that an exponential blowup for the resulting weak ω-DFA is unavoidable.

4 Finite State Strategies for Pushdown Games

In this section, we are going to generalize the regularity problem to pushdown games. Testing for regularity in this setting means to ask for the existence of a winning finite state strategy. Let us first reconsider the weak classification game from Definition 2 which connects ω-languages and games in the context of regularity testing. We show that having a finite state representation for an ω-language naturally extends to having a finite state winning strategies.

Lemma 16. *Let $\mathcal{A}$ be a weak ω-DPDA with color set C. $L_\omega(\mathcal{A}) \subseteq \Sigma^\omega$ is regular iff Player 0 can win the weak classification game $\mathcal{G}_{\mathcal{A},C}$ with a finite state strategy.*

Proof. Let $\mathcal{A}$ and C be as above. The ω-language $L_\omega(\mathcal{A})$ is regular iff it is recognized by a weak ω-DFA $\mathcal{A}'$ with colors C according to Lemma 3. Such a weak ω-DFA naturally corresponds to a winning FSS for Player 0 in $\mathcal{G}_{\mathcal{A},C}$. □

In the rest of this section, we study the decision problem whether Player 0 can win a weak PDG with an FSS. We give a trivial answer in case of games with reachability condition but a negative answer for safety condition.

Lemma 17. *For a reachability PDG, Player 0 has a winning finite state strategy if he can win.*

Proof. Let $\mathcal{G} = (\mathcal{M}, \tau, Q_0)$ be a weak PDG with $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$, colors $\tau(Q) = \{0, 1\}$, and f a winning strategy for Player 0. We consider all possible play prefixes where Player 0 plays according to f until a state of color 0 is reached. Such a state exists on each play since f is winning for Player 0. When we arrange these play prefixes as a tree, then it is a finitely branching tree in which each branch is finite. According to König's Lemma, the tree must be finite. Using the nodes of this tree as states of a finite automaton yields a finite state winning strategy. □

Lemma 17 implies the following since PDGs are effectively determined [19].

Corollary 18. *For a reachability PDG, it is decidable whether Player 0 has a winning finite state strategy.*

This problem is not symmetric, meaning that it is undecidable from the perspective of Player 1, which is about having a winning FSS for a safety condition. To this end, we encode the run of a *2-register machine (2RM)* as a pushdown game. A 2RM can be seen as an input-free deterministic machine that is equipped with 2 counters which can be increased, decreased and tested for zero. The halting problem is undecidable for this machine model as it can encode Turing machines. For our next proof, we consider a similar problem which is to decide whether the unique run of a 2RM is *ultimately periodic (UP)*, i.e., whether the state sequence starting from the initial configuration forms an infinite word uv^ω where u, v are nonempty finite state sequences.

Lemma 19. *It is undecidable whether the run of a 2RM is ultimately periodic.*

Theorem 20. *For a safety 1CG, it is undecidable whether Player 0 has a winning finite state strategy.*

Proof (Sketch). We use Lemma 19 and construct for a given 2RM M a safety 1CG $\mathcal{G}$ such that Player 0 can win with an FSS iff M has a UP run. The main idea of $\mathcal{G}$ is divided into two phases. First, Player 1 increases and decreases the counter to an arbitrary value. Whether at this point the value is zero or not determines which register of M will be simulated by the counter of $\mathcal{G}$ in the second phase. There Player 0 has to play an infinite transition sequence of M that is correct just according to the simulated register. Thus, a play is won by Player 0 iff Player 1 never leaves the first phase, or the second phase is reached and Player 0 can give an infinite run of M which simulates the according register correctly. This can be modeled by a safety winning condition where only an incorrect simulation in the second phase leads to an unsafe state.

Obviously, Player 0 has a winning strategy. But here we are interested in a winning FSS for Player 0. If he has a winning FSS f, then its limited memory cannot remember after the first phase whether the counter is zero or not, i.e., which register will be simulated by $\mathcal{G}$ in the second phase. But independent from that information, f is winning in the second phase and hence the transition sequence that f produces must be correct in both cases. This means that the sequence is indeed the unique run of M, and since it can be represented by the FSS f, it must be UP. For the inverse, assume that the run of M is UP. Then it can be represented by a finite state machine. Hence, a winning FSS for Player 0 is to ignore the first phase and to play in the second phase the run of M which is correct with respect to both registers. □

The undecidability of that problem can be shown in a similar way for visibly PDG, i.e., where the type of the stack operation is determined by the input symbol played in the game [1]. Due to this new restriction, the construction needs additional stack symbols to make an FSS of Player 0 not see what is happening on the stack.

Theorem 21. *For a safety visibly PDG, it is undecidable whether Player 0 has a winning finite state strategy.*

5 Conclusions

In Section 3, we considered weak pushdown ω-automata. We established a normal form that each weak pushdown ω-automaton can be effectively converted to, and revealed its close relation to languages of finite words. This allowed us to lift known decision procedures for equivalence [14] and regularity [18] from the case of finite words to ω-automata. Our regularity test can be performed in triply exponential time. The worst case size of an equivalent ω-DFA is between singly and doubly exponential. Especially the upper bound for ω-languages coincides with the one given by Valiant [18] for languages of finite words. He further gave singly exponential bounds for some restricted types of pushdown automata (like

ε-free, 1-counter). It is open whether they can be used to improve the test on respective types of ω-automata. Finally, the decidability of the regularity problem for (non-weak) pushdown ω-automata, as formulated in [6], remains open.

We dedicated Section 4 to an extension of the regularity problem for pushdown games, which asks for the existence of a finite state winning strategy. Beside a simple decidability result in the case of reachability winning conditions, we showed in the main result of that section the undecidability for safety conditions. The latter result extends to more complicated winning conditions of course. The decidability remains open in the case of visibly one-counter games.

Acknowledgements. With thanks to Wladimir Fridman for fruitful discussions.

References

[1] Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC, pp. 202–211 (2004)

[2] Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press (2008)

[3] Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Transactions of the AMS 138, 295–311 (1969)

[4] Cachat, T.: Symbolic Strategy Synthesis for Games on Pushdown Graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 704–715. Springer, Heidelberg (2002)

[5] Chandra, A.K., Kozen, D., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)

[6] Cohen, R.S., Gold, A.Y.: Omega-computations on deterministic pushdown machines. JCSS 16(3), 275–300 (1978)

[7] Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient Algorithms for Model Checking Pushdown Systems. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 232–247. Springer, Heidelberg (2000)

[8] Hague, M., Ong, C.-H.L.: Winning Regions of Pushdown Parity Games: A Saturation Method. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 384–398. Springer, Heidelberg (2009)

[9] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)

[10] Löding, C.: Efficient minimization of deterministic weak omega-automata. Information Processing Letters 79(3), 105–109 (2001)

[11] Perrin, D., Pin, J.-É.: Infinite words. Pure and Applied Mathematics, vol. 141. Elsevier (2004)

[12] Segoufin, L., Sirangelo, C.: Constant-Memory Validation of Streaming XML Documents Against DTDs. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 299–313. Springer, Heidelberg (2006)

[13] Segoufin, L., Vianu, V.: Validating streaming XML documents. In: PODS, pp. 53–64 (2002)

[14] Sénizergues, G.: L(A)=L(B)? decidability results from complete formal systems. Theor. Comput. Sci. 251(1-2), 1–166 (2001)

[15] Staiger, L.: Finite-state ω-languages. JCSS 27(3), 434–448 (1983)

[16] Stearns, R.E.: A regularity test for pushdown machines. Information and Control 11(3), 323–340 (1967)

[17] Straubing, H.: Finite Automata, Formal Logic, and Circuit Complexity. Birkhäuser, Basel (1994)

[18] Valiant, L.G.: Regularity and related problems for deterministic pushdown automata. J. ACM 22(1), 1–10 (1975)

[19] Walukiewicz, I.: Pushdown processes: Games and model checking. Information and Computation 164(2), 234–263 (2001)

Computational Aspects of Cellular Automata on Countable Sofic Shifts*

Ville Salo[1] and Ilkka Törmä[2]

[1] TUCS – Turku Centre for Computer Science, Finland,
University of Turku, Finland
vosalo@utu.fi

[2] University of Turku, Finland
iatorm@utu.fi

Abstract. We investigate the computational properties of cellular automata on countable (equivalently, zero entropy) sofic shifts with an emphasis on nilpotency, periodicity, and asymptotic behavior. As a tool for proving decidability results, we prove the Starfleet Lemma, which is of independent interest. We present computational results including the decidability of nilpotency and periodicity, the undecidability of stability of the limit set, and the existence of a Π^0_1-complete limit set and a Σ^0_3-complete asymptotic set.

Keywords: cellular automata, subshifts, nontransitivity, undecidability.

1 Introduction

Symbolic dynamics is often viewed from the perspective of coding, and then infinite words represent information flowing to the left in the orbit given by the shift action. From this perspective, positive dynamical entropy is crucial, and transitivity of the subshift is a very natural assumption. In the case of a sofic shift, transitivity essentially means that the behavior of the medium stays the same no matter what has been sent sofar. However, from a purely mathematical perspective, zero entropy, or countable, sofic shifts are quite interesting, consisting of a simple periodic background pattern broken by a finite number of local disturbances. They lie in the other extremity of the spectrum of sofic shifts, where almost no information can be sent. Still, as far as we know, the conjugacy problem is still open even in the case of countable SFTs.

Our point of view is that of investigating the computational capabilities of cellular automata running on a countable sofic shift. Such a cellular automaton can be thought of as a kind of counter machine, with the distances between the local disturbances representing the counter values. As a tool for proving decidability results for such systems, we prove the Starfleet Lemma, which provides insight into the precise dynamics the cellular automata, and is of independent interest.

* Research supported by the Academy of Finland Grant 131558.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 777–788, 2012.

We present a variety of computational results: We prove the decidability of nilpotency and periodicity and that limit sets and asymptotic sets always lie in Π_1^0 and Σ_3^0 in the arithmetical hierarchy, respectively. Further, we prove that limit sets and asymptotic sets in fact capture the respective classes by finding complete sets for them. The examples proving completeness are based on a rather direct simulation of counter machines by CA, and we also obtain many undecidability results using this idea.

2 Definitions

Let S be a finite set, called the *state set* or *alphabet*. The set $S^{\mathbb{Z}}$ of bi-infinite state sequences, or *configurations*, is called the *full shift on* S. If $x \in S^{\mathbb{Z}}$, then we denote by x_i the ith coordinate of x, and we adopt the shorthand notation $x_{[i,j]} = x_i x_{i+1} \ldots x_j$. If $w \in S^*$, we denote $w \sqsubset x$, and say that w *occurs in* x, if $w = x_{[i,i+|w|-1]}$ for some i. For words $t, u, v, w \in S^*$, the notation ${}^\infty tu.vw^\infty$ has the intuitive meaning, with the word v starting at coordinate 0. The dot can be omitted if the position of the words is irrelevant. Two elements $x, y \in S^{\mathbb{Z}}$ are *right (left) asymptotic* if $x_i = y_i$ for all sufficiently large (small) i.

We define a metric d on the full shift by $d(x, y) = \inf\{2^{-n} \mid x_{[-n,n]} = y_{[-n,n]}\}$. The topology defined by d makes $S^{\mathbb{Z}}$ a compact metric space. We define the *shift map* $\sigma : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ by $\sigma(x)_i = x_{i+1}$. Clearly σ is a homeomorphism from the full shift to itself.

A *subshift* is a closed subset X of the full shift with the property $\sigma(X) = X$. Alternatively, a subshift is defined by a set $F \subset S^*$ of *forbidden words* as the set $\mathcal{X}_F = \{x \in S^{\mathbb{Z}} \mid \forall w \in F : w \not\sqsubset x\}$. If F is finite, then $\mathcal{X}_F$ is *of finite type*, and if F is regular, $\mathcal{X}_F$ is *sofic*. We define $\mathcal{B}_k(X) = \{w \in \Sigma^k \mid \exists x \in X : w \sqsubset x\}$ as the set of words of length k occurring in X, and define the *language* of X as $\mathcal{B}(X) = \bigcup_{k \in \mathbb{N}} \mathcal{B}_k(X)$. The language of a subshift determines it [1], so we may also denote $X = \mathcal{B}^{-1}(L)$, if $\mathcal{B}(X)$ is the set of factors of L. We denote by σ_X the restriction of σ to X.

Given a directed graph $G = (V, E)$, we define its *edge shift*, a subshift of $E^{\mathbb{Z}}$, as the set of bi-infinite paths on its edges. Every sofic shift X is equal to the set of labels of bi-infinite paths on a directed graph with labeled edges, and the edge shift of the (essentially unique) deterministic graph with the least number of edges is called the *minimal Shannon cover* of X.

Let $X \subset S^{\mathbb{Z}}$ be a subshift. A *cellular automaton* is a continuous function $c : X \to X$ with the property $c \circ \sigma_X = \sigma_X \circ c$. Equivalently, cellular automata can be defined by *local functions* $C : \mathcal{B}_{2r+1}(X) \to \mathcal{B}_1(X)$ for some $r \in \mathbb{N}$ such that $c(x)_i = C(x_{[i-r,i+r]})$. The smallest such r is called the *radius* of c. The *limit set* of c is $\bigcap_{n \in \mathbb{N}} c^n(X)$, and c is *stable* if the limit set is equal to some $c^n(X)$. A state 0 is *quiescent* for c if $c({}^\infty 0^\infty) = {}^\infty 0^\infty$. A CA c on X is *weakly nilpotent* if for all $x \in X$ we have an n such that $c^n(x) = {}^\infty 0^\infty$, and *nilpotent* if the n are uniformly bounded. Also, c is *weakly periodic* if for all $x \in X$ we have an n such that $c^n(x) = x$, and *periodic* if the n are uniformly bounded. A *spaceship of* c is a configuration $x \in X$ such that $c^n(x) = \sigma^i(x)$ for some $n \in \mathbb{N}$ and $i \in \mathbb{Z}$. We say the spaceship is *nontrivial* if it is not spatially periodic.

Let $k \in \mathbb{N}$. A *k-counter machine* is a quintuple $M = (\Sigma, k, \delta, q_0, q_f)$, where Σ is a finite *state set*, $q_0, q_f \in \Sigma$ the *initial and final states* and $\delta : (\Sigma - \{q_f\}) \rightarrow [1,k] \times \Sigma^2 \cup [1,k] \times \{\uparrow, \downarrow\} \times \Sigma$ the *transition function*. A *configuration of M* is an element of $\mathbb{N}^k \times \Sigma$, with the interpretation of $(n_1, \ldots, n_k, s)$ being that the machine is in state s with counter values $n_1, \ldots, n_k$. The machine operates in steps as directed by δ. The case $\delta(s) = (i, r, t) \in [1,k] \times \Sigma^2$ means that if M is in state s, then it will change its state to r if $n_i = 0$, and otherwise to t. The case $\delta(s) = (i, d, r) \in [1,k] \times \{\uparrow, \downarrow\} \times \Sigma$ means that M increments or decrements n_i by 1, depending on d (but never decrementing it below 0), and then goes to state r. The *language* $L(M)$ of M is the set of $n \in \mathbb{N}$ such that $(n, 0, \ldots, 0, q_0)$ eventually reaches a configuration with state q_f. The machine M is *reversible* if every configuration can be reached in one step from at most one other configuration. Note that a reversible counter machine need not be bijective, only injective.

Let ϕ be a formula in first-order arithmetic. If ϕ contains only bounded quantifiers, then we say ϕ is Σ^0_0 and Π^0_0. For all $n > 0$, we say ϕ is Σ^0_n if it is equivalent to a formula of the form $\exists k : \psi$ where ψ is Π^0_{n-1}, and ϕ is Π^0_n, if it is equivalent to a formula of the form $\forall k : \psi$ where ψ is Σ^0_{n-1}. This classification is called the *arithmetical hierarchy*. A subset X of $\mathbb{N}$ is Σ^0_n or Π^0_n, if $X = \{x \in \mathbb{N} \mid \phi(x)\}$ for some ϕ with the corresponding classification. It is known that the class Σ^0_1 consists of exactly the recursively enumerable subsets of $\mathbb{N}$, and Π^0_1 of their complements.

A subset $X \subset \mathbb{N}$ is *many-one reducible* (or simply *reducible*) to another set $Y \subset \mathbb{N}$, if there exists a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that $x \in X$ iff $f(x) \in Y$. If every set in a class $\mathcal{C}$ is reducible to X, then X is said to be *$\mathcal{C}$-hard*. If, in addition, X is in $\mathcal{C}$, then X is *$\mathcal{C}$-complete*.

3 Basics

We remark here, and it would not be hard to prove either, that a sofic shift is countable if and only if the cycles of its Shannon cover are disjoint. Thus the configurations of a countable sofic shift consist of arbitrarily long periodic patterns (corresponding to the cycles of its Shannon cover), separated by finite period-breaking patterns (the transitions between cycles). In particular, there are a finite number of periodic configurations and a global upper bound on the number of transitions in a single configuration. Furthermore, a sofic shift is easily seen to be countable if and only if it has zero topological entropy with respect to the shift map, but we will not elaborate on this here (see [1, Chapter 4] for more information).

We now prove a fundamental result concerning the dynamics of cellular automata on countable sofic shifts, which is very useful for decidability results. For this, we need a canonical representation for configurations of the shift, given by the following lemma.

Lemma 1. *Let X be a countable sofic shift. Then there exists a finite set T of tuples of words in $\mathcal{B}(X)$ such that every point $x \in X$ is representable as $x = {}^{\infty}u_0 v_1 u_1^{n_1} \cdots u_{m-1}^{n_{m-1}} v_m u_m^{\infty}$ for a unique $t = (u_0, \dots, u_m, v_1, \dots, v_m) \in T$.*

Proof. Let p be a common period for all the periodic points of X, and let $U = \{u \in \mathcal{B}_p(X) \mid {}^{\infty}u^{\infty} \in X\}$. For all $x \in X$ we add a tuple $T(x) \in T$ corresponding to x as follows. First, there is a unique $u_0 \in U$ such that x is of the form ${}^{\infty}u_0 a y$ for some letter $a \in S - \{(u_0)_1\}$ and $y \in S^{\mathbb{N}}$. This follows from the disjointness of the cycles in the Shannon cover of X. Let the coordinate of a be i_1.

We inductively define i_k as the first coordinate greater than i_{k-1} such that $u_k := x_{[i_k - p, i_k - 1]} \in U$ and $x_{i_k} \neq (u_k)_1$. When such a coordinate can no longer be found, we are in the right periodic tail of x. Finally, the last u_m is chosen as we would choose u_0 for x^R, the reversal of x. The v_k are then the words occurring in between the u_k-periodic patterns, and we set $T(x) = (u_0, \dots, u_m, v_1, \dots, v_m)$.

This representation is now unique, since the parsing process is deterministic from left to right. □

We continue to write $T(x)$ for the unique tuple $(u_0, \dots, u_m, v_1, \dots, v_m)$ given by the previous lemma for $x \in X$, and $M(x)$ for the tuple $(n_1, \dots, n_{m-1})$.

Lemma 2 (Starfleet Lemma). *If c is a cellular automaton on a countable sofic shift X, then for all $x \in X$ there exists a tuple $t = (r_0, \dots, r_\ell, s_1, \dots, s_\ell) \in \mathcal{B}(X)^{2\ell+1}$ (not necessarily in T) such that*

- *for all $N \in \mathbb{N}$, there exist $n \in \mathbb{N}$ and $n_1, \dots, n_{\ell-1} \geq N$ such that*

$$c^n(x) = {}^{\infty}r_0 s_1 r_1^{n_1} s_2 \cdots s_{\ell-1} r_{\ell-1}^{n_{\ell-1}} s_\ell r_\ell^{\infty}$$

  modulo a power of the shift, and
- *for all $i \in [1, \ell]$, the configuration ${}^{\infty}r_{i-1} s_i r_i^{\infty}$ is a nontrivial spaceship for c.*

We call a tuple t as above a *starfleet for x.*

Proof. Let $x \in X$. Define $U_x = \{(T(c^n(x)), M(c^n(x))) \mid n \in \mathbb{N}\}$, and for all $t \in T$, let $W_x(t)$ be the set $\{\tau \mid (t, \tau) \in U_x\}$. Let $t = (u_0, \dots, u_m, v_1, \dots, v_m) \in T$, and let $C_x(t)$ be the closure of the set $W_x(t)$ in $\bar{\mathbb{N}}^{m-1}$, where $\bar{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$ is the one-point compactification of $\mathbb{N}$.

If $C_x(t)$ is finite for all $t \in T$, then x must evolve into a spaceship, and the requirements of the lemma clearly hold. If this is not the case, let $t = (u_0, \dots, u_m, v_1, \dots, v_m) \in T$ and $\tau = (n_1, \dots, n_{m-1}) \in C_x(t)$ be such that τ contains a maximal number of infinite coordinates. Let $(k_1, \dots, k_{\ell-1})$ be the infinite coordinates of τ, let $k_0 = 0$ and $k_\ell = m$, and consider the words

$$w_i = v_{k_{i-1}+1} u_{k_{i-1}+1}^{n_{k_{i-1}+1}} \cdots u_{k_i - 1}^{n_{k_i - 1}} v_{k_i},$$

for $i = 1, \dots, \ell$.

Clearly, by the maximality of ℓ, each ${}^{\infty}u_{k_{i-1}} w_i u_{k_i}^{\infty}$ will evolve into a nontrivial spaceship ${}^{\infty}r_{i-1} s_i r_i^{\infty}$ in some p steps. Let $x_1, x_2, \dots$ be the subsequence of the orbit of x such that $T(x_i) = t$ and $M(x_i)$ converges to τ. Then, the subsequence $x'_i = c^p(x_i)$ shows that $(r_0, \dots, r_\ell, s_1, \dots s_\ell)$ has the required properties. □

For CA on a mixing sofic shift, it is well-known that injectivity implies surjectivity [1]. However, in general, surjectivity and injectivity do not imply each other even in a countable SFT. The following example shows this, and will also turn out useful later on.

Example 1. Let $X = \mathcal{B}^{-1}(0^*1^*2^*)$ and let f be a cellular automaton that expands the pattern of 1's to both directions by one. Clearly, such a CA is injective on X, but it is not surjective. Similarly, the CA g which removes a single 1 from each end is surjective, but not injective (in fact, not even preinjective, that is, two distinct left and right asymptotic configurations can have the same image).

Of course, a CA that is surjective and injective is still reversible. Furthermore, the usual decidability results hold also in the general case: Surjectivity, injectivity and reversibility of cellular automata are decidable properties on all sofic shifts, which is easily seen using standard techniques of symbolic dynamics.

4 Nilpotency and Periodicity

In this section, we prove that nilpotency and periodicity are decidable properties for cellular automata on countable sofic shifts. The decidability of nilpotency is a rather direct application of the Starfleet Lemma and the following result.

Lemma 3. *Let X be a subshift. Then a cellular automaton c on X is nilpotent if and only if it is weakly nilpotent.*

Proposition 1. *For cellular automata on countable sofic shifts, nilpotency is decidable.*

Proof. We will prove that one of the following cases holds for a CA c on a countable sofic shift X: either c is nilpotent, it is not nilpotent on periodic configurations, or X contains a nontrivial spaceship for c. Since the latter cases imply non-nilpotency and all three are semi-decidable, this proves the proposition.

Assume that c is not nilpotent, but is nilpotent on the finitely many periodic configurations of X, and let ${}^\infty 0^\infty$ be their limit. We will show that X contains a nontrivial spaceship for c. By Lemma 3, c is not even weakly nilpotent, so there exists a configuration $x \in X$ with $c^n(x) \neq {}^\infty 0^\infty$ for all n. Let $(r_0, \ldots, r_\ell, s_1, \ldots, s_\ell) \in \mathcal{B}(X)^{2\ell+1}$ be a starfleet for x, which must be nontrivial: $s_i \notin 0^*$ for some i. But now ${}^\infty r_{i-1} s_i r_i^\infty$ is a nontrivial spaceship for c. □

Note that this is not true in general: It is proved in [2] that on the full shift, nilpotency is undecidable for one-dimensional cellular automata. In fact, there is an analogue of Rice's theorem on the limit sets of CA on the full shift [3], implying that all nontrivial properties of the limit set are undecidable. An interesting question is whether some weaker form of Rice's theorem holds on countable sofic shifts, since Theorem 2 implies that individual limit sets can have very complicated structure.

The decidability of periodicity also follows from the Starfleet Lemma.

Lemma 4. *A cellular automaton c on an arbitrary subshift $X \subset S^{\mathbb{Z}}$ is periodic iff it is weakly periodic.*

Proof. We only need to prove that weak periodicity implies periodicity, so let c be weakly periodic. Then its *two-way trace subshift*

$$\{y \in S^{\mathbb{Z}} \mid \exists (x^i)_{i \in \mathbb{Z}} \in X^{\mathbb{Z}} : \forall i \in \mathbb{Z} : x^{i+1} = c(x^i) \wedge y_i = x^i_0\}$$

contains only periodic points, and thus must be finite [4]. Then c is periodic with period the least common multiple of this finite set of periods. □

Proposition 2. *For cellular automata on countable sofic shifts, periodicity is decidable.*

Proof. We will prove that one of the following cases holds for a CA c on a countable sofic shift X: either c is periodic, c is noninjective, or X contains a nonperiodic spaceship for c. Since the latter cases imply nonperiodicity and all three are semi-decidable, this proves the proposition.

Suppose on the contrary that c is nonperiodic (hence not weakly periodic by Lemma 4) and injective, and all spaceships are periodic. Now let $x \in X$ be arbitrary, and let $(r_0, \dots, r_\ell, s_1, \dots, s_\ell)$ be a starfleet for it. Since the configurations ${}^\infty r_{i-1} s_i r_i^\infty$ are spaceships, they must be periodic, so $\ell = 1$. Now $c^n(x)$ is a spaceship for large enough n, and by injectivity of c, x is a spaceship itself, hence periodic. This is a contradiction, since c was not weakly periodic. □

We remark that the decidability proofs of nilpotency and periodicity both involve three semi-decidable properties, of which every cellular automaton must possess at least one. In addition to proving the propositions, these triplets are interesting in themselves, shedding light on the possible dynamics of cellular automata on countable sofic shifts.

5 Computation and Limit Sets

In this section, we focus on the limit sets of cellular automata and their computational properties. We start by defining a natural tool for proving undecidability results, namely, a cellular automaton which simulates a counter machine. Since the limit set is exactly the set of configurations with an infinite chain of preimages, it is natural to run a counter machine in reverse.

Theorem 1 ([5]). *For any k-counter machine $M = (\Sigma, k, \delta, q_0, q_f)$ there exists a reversible $(2k+2)$-counter machine $M' = (\Sigma', 2k+2, \delta', p_0, p_f)$ such that*

$$(m_1, \dots, m_k, q_0) \Rightarrow^*_M (n_1, \dots, n_k, q_f)$$

if and only if

$$(m_1, \dots, m_k, 0, 0, 0, \dots, 0, p_0) \Rightarrow^*_{M'} (m_1, \dots, m_k, 0, 0, n_1, \dots, n_k, p_f).$$

Let $M = (\Sigma, k, \delta, q_0, q_f)$ be a counter machine. We construct a countable SFT X_M and a CA c_M on X_M simulating M in a specific way.

Denote $\Sigma' = \Sigma \times \{\leftarrow, \rightarrow\}$. The subshift X_M is defined as the subset of

$$\mathcal{B}^{-1}(a^* \Sigma' b^*) \times \prod_{i \in [1,k] \cup \{\#\}} \mathcal{B}^{-1}(a^* i b^*)$$

where none of the symbols $[1, k]$ can occur to the left of #, and not all symbols $[1, k] \cup \#$ can occur on the same side of $s \in \Sigma'$. A configuration of X_M is *good* if it contains some $s' \in \Sigma'$, the symbol # and all symbols $i \in [1, k]$.

The automaton c_M simulates M as follows. The value of a counter i is coded as its distance from the symbol #. Each state $(s, \rightarrow) \in \Sigma'$ starts on #, where the automaton can immediately check whether any counter has value 0. If needed, the state travels to the right until it encounters the counter symbol. It shifts the symbol one step to the desired direction, flips its own arrow, returns to the # and enters a new state as determined by δ. If x is a configuration of M, we denote by $X_M(x)$ the corresponding family of configurations of X_M (the configuration with # placed in the origin, and all its shifts).

If M is reversible, we may add a *direction bit* to each state and have c_M simulate M either forward or backward, flipping the bit whenever an image or preimage does not exist. We denote this augmented automaton by $\bar{c}_M$. Note that this is possible since a CA can, in one step, check which counters have value 0, when the state lies on the #-symbol, even if the reverse function cannot actually be computed by a counter machine. The notation $X_M^{\rightarrow}(x)$ stands for the forward and $X_M^{\leftarrow}(x)$ for the backward configuration family.

When this basic construction is used in proofs, it will be modified as needed.

It is easy to see that the language of a limit set of a CA is always Π_1^0, and the following complementary theorem holds.

Theorem 2. *There exists a CA $c : X \rightarrow X$ on a countable SFT X such that the limit set of c is Π_1^0-complete.*

Proof. Let M be a reversible counter machine whose language is Σ_1^0-complete with the property that the initial configuration $(n, 0, \ldots, 0, q_0)$ has no preimage for any n, and if $n \notin L(M)$, then M does not halt when started on this configuration. Such a machine can be constructed using Theorem 1 and its proof from [5]. We modify $\bar{c}_M$ to start filling the space with a new spreading state when an accepting state is reached in a forward state.

Now we claim that a forward CA configuration corresponding to

$$x_n = (n, 0, \ldots, 0, q_0)$$

is in the limit set of $\bar{c}_M$ if and only if $n \notin L(M)$. It is clear that if $n \notin L(M)$, then an infinite chain of preimages is obtained by running M backwards towards the initial state. Consider the case $n \in L(M)$. Since M is reversible and $\bar{c}_M$ simulates it on good configurations, the only possible preimage chain for an element of $X_M^{\rightarrow}(x_n)$ under $\bar{c}_M$ could be obtained by running M back and forth between the initial and accepting states. But this is not possible, since we introduced a spreading state, and adding a spreading state clearly cannot add any new preimages for good configurations. □

Theorem 3. *It is undecidable whether a CA on a countable SFT is stable.*

Proof. Given a reversible counter machine M, it easily follows from Theorem 1 that it is undecidable whether M halts on $x = (0, \dots, 0, q_0)$. Modify $\bar{c}_M$ so that the configurations in $X_{\overleftarrow{M}}(x)$ become fixed points, but no other configuration is affected. We now claim that the modified $\bar{c}_M$ is stable iff M halts from x.

Suppose first that M does not halt from x. Then every point in $\bar{c}_M^n(X_{\overrightarrow{M}}(x))$ has a single preimage chain of length n ending in $X_{\overrightarrow{M}}(x)$. Thus $\bar{c}_M$ is unstable.

Suppose then that M halts from x. Then $X_{\overrightarrow{M}}(x)$ eventually evolves into $X_{\overrightarrow{M}}(y)$ for a final counter machine configuration y. From there, $\bar{c}_M$ will run backwards to $X_{\overleftarrow{M}}(x)$, which consists of fixed points. Thus there are (up to a shift) only a finite number of points in $Y = \bigcup_{n \in \mathbb{N}} \bar{c}_M^n(X_{\overrightarrow{M}}(x))$ without an infinite preimage chain. Furthermore, every point of $X_M - Y$ has an infinite chain of preimages, and thus $\bar{c}_M$ is stable. □

6 Asymptotic Sets

The limit set is not the only notion corresponding to 'where points eventually go'. Another such concept, studied at least in [6], is the asymptotic set. We show that this idea makes sense also in the countable sofic case, and is in some sense stronger than the concept of limit set.

Definition 1. *The* asymptotic set *of CA $c : X \to X$ is*

$$\bigcup_{x \in X} \bigcap_{J \in \mathbb{N}} \overline{\{c^n(x) \mid n \geq J\}}$$

A configuration $x \in X$ lies in the asymptotic set iff there exists another configuration $y \in X$ and a subsequence of the orbit $(c^n(y))_{n \in \mathbb{N}}$ which converges to x. Note that the asymptotic set contains all temporally periodic points of c, but not necessarily all the spaceships, unlike the limit set. Asymptotic sets have much stronger computational capabilities than limit sets.

Lemma 5. *The asymptotic set Y of a CA c on a countable sofic shift X is Σ^0_3.*

Proof. Given a word w, it is clearly (by its form) in Σ^0_3 to check that

$$\exists x \in X : \forall n : \exists m > n : c^m(x)_{[1,|w|]} = w,$$

which is equivalent to $w \in \mathcal{B}(Y)$. Note that the values x of the first quantifier can easily be enumerated by a Turing machine. □

Theorem 4. *There exists a countable SFT X and a CA $f : X \to X$ such that the asymptotic set of f is Σ^0_3-complete.*

Proof. It is easy to see that there exists a recursive set L such that solving

$$\exists r : \forall \ell : \exists m : (r, \ell, m, w) \in L$$

for given $w \in \mathbb{N}$ is Σ_3^0-complete. We say that such an w is a *solution to* L. We will many-one reduce any such set to the language of the asymptotic set of a CA.

Let M be an always halting counter machine for L which never re-enters the initial state. We construct a counter machine M' with state set $\{s_1, s_2, \ldots, s_p\}$, counters $C_w, C_r, C_\ell, C'_\ell, C_m$, and a suitable set of auxiliary counters, having the property that started from configuration $(s_1, w, r, i_1, \ldots, i_h)$, M' enters the state s_1 infinitely many times if and only if w is a solution to L for that choice of r, and the i_j are arbitrary. The counter C_w will always contain the value w and is never modified. The counters C_r, C_ℓ, C'_ℓ and C_m in M' will play the role of quantifiers, with C_r containing the guess r. The idea of the construction is that C'_ℓ will loop through the values $[1, n]$ for larger and larger n, which is stored in C_ℓ, and at every value ℓ, C_m is used to search for an m such that $(r, \ell, m, w) \in L$. Such an event is followed by setting all counters to 0, except C_w, C_r, C_ℓ and C'_ℓ, and visiting the state s_1. It follows that the state s_1 is visited infinitely many times if and only if r was a correct guess for w.

We wish to have the central pattern of the configuration corresponding to $(s_1, w, \infty, \ldots, \infty)$ in the asymptotic set of the corresponding CA if and only if w is a solution to L. Infinite values in counters during the simulation are handled by simply making the machine head check that all counter values are finite before entering s_1. We also have the problem that while entering s_1 infinitely many times indeed characterizes the solutions of L, the values of other counters might be visible in the asymptotic set. We thus modify all counters except C_w to store their data in the form $2^n \cdot n'$, where n' is odd and n is the useful piece of data, and increase n' by 2 every time M enters the state s_1. □

Corollary 1. *There exists a CA c on the full shift such that the asymptotic set of c is Σ_3^0-complete.*

Proof. Let X and c be given by Theorem 4. We embed X in a full shift and add a spreading state t for c which appears whenever a neighborhood is forbidden in X. Then we may use the same reduction as before, since points outside X only contribute words in t^* to the language of the asymptotic set. □

Starting from a fixed sofic shift, asymptotic sets do not necessarily form a larger class than limit sets:

Proposition 3. *There exists a countable sofic shift X and a CA $c : X \to X$ such that the limit set of c is not the asymptotic set of any CA on X.*

Proof. Let

$$X = \mathcal{B}^{-1}(0^*\ell 0^*\#0^*r0^* + 0^*\ell' 0^*\#0^*r'0^*)$$

and c the CA on X which moves ℓ and r towards #, and moves ℓ' and r' away from #, changing $\ell\#r$ to $\ell'\#r'$, but $\ell\#0$ and $0\#r$ to $0\#0$. It is clear that the limit set is

$$Y = \mathcal{B}^{-1}(\bigcup_{n \in \mathbb{N}} (0^*\ell 0^* \# 0^* r 0^* + 0^* \ell' 0^n \# 0^n r' 0^*)).$$

We claim that no CA e on X has Y as its asymptotic set. Assume on the contrary that this is possible, and let e have radius R. In general, it is clearly true for asymptotic sets that for all N, the $e^n(x)_{[-N,N]}$ must be a word of Y for all x and sufficiently large n. In particular, words of Y must map to words of Y. Also, points in Y have preimages in Y.

Since configurations of the form $y(M, M') = {}^\infty 0\ell 0^M \# 0^{M'} r 0^\infty$ appear in the asymptotic set with no restriction on M and M', but the point $z(M, M') = {}^\infty 0\ell' 0^M \# 0^{M'} r' 0^\infty$ only appears for $M = M'$, a simple case analysis shows that for large M and M', such points map to points of the same form.

Since $z(M, M)$ is isolated in X, it must be e-periodic, and since it is in the asymptotic set, it must map to a point of the form $z(N, N)$. If for some M, a point of the form $z(N, N)$ with $N \leq 2R - 1$ never appeared in the orbit, we would also find an e-periodic point of the form $z(M, M')$ with $M \neq M'$. Therefore, there exists a point $z(N, N)$ with $N \leq 2R - 1$ such that $z(M, M)$ appears in its orbit for arbitrarily large M. But this is a contradiction, since $z(N, N)$ must be e-periodic.

However, if the subshift is not fixed, we have the following realization theorem. Note that this is not stronger than Theorem 4, since it only implies the existence of a sofic shift, and not an SFT.

Theorem 5. *Let X be a countable sofic shift and let Y be a Σ^0_3 subshift of X. Then there exists a countable sofic shift $Z \supset X$ and a CA $c : Z \to Z$ such that the asymptotic set of c from Z is exactly Y.*

Proof. The subshift Z has the same periodic points as Y and the cellular automaton c having Y as an asymptotic set from Z behaves as the identity map on the periodic points. Let p be a common period for the periodic points of Z.

If x and y are periodic, and there is a unique point z that is left asymptotic to x and right asymptotic to y, then c also behaves as the identity map on z. For all pairs of periodic points without this property, we construct a subshift $Z(x, y)$ such that starting from $Z(x, y)$, the automaton c produces exactly the points of Y left and right asymptotic to x and y, respectively, in the asymptotic set.

Let x be u-periodic and y be v-periodic for $|u| = |v| = p$. The general form of a 'good' point in $Z(x, y)$ is

$${}^\infty u M_1 u^* w v^* M_2 v^\infty,$$

where w is called a *message* moving either left or right, such that ${}^\infty u w v^\infty$ appears in X, and $u^* w v^*$ is called the *periodic medium.* The words M_1 and M_2 encode counter machines. Since there are at least two points left and right asymptotic to x and y, respectively, we may choose a nonperiodic word w_{ok} such that ${}^\infty u w_{ok} v^\infty$

is an isolated point in X. Between a pattern of u's and v's, w_{ok} is shifted a multiple of p steps to the right, and all other words are shifted some multiple of p steps to the left. Since there are only finitely many different choices of x and y, it is easy to handle the finite amount of words w_{ok} separately and move them in a different direction than all others (since these islands are maximal).

By the assumption on Y, the set of good words w that occur between ${}^{\infty}u$ and v^{∞} must be given by the solutions w to

$$\exists k : \forall l : \exists m : (k, l, m, w) \in L$$

for a recursive language L.

The machines M_1 and M_2 have to be encoded slightly differently than usual, so that large values in counters do not change the set of periodic points of Z from those of Y. The finitely many counter markers, however, can be over a separate alphabet altogether. In the evolution of c, the machines M_1 and M_2 move slowly away from each other, so that they cannot be seen in the asymptotic set. The 'base points' of counters (where counters become zero) of M_1 and M_2 are on the side of the periodic medium between them. This makes the sending of messages more intuitive, and makes it clearer that the machines actually disappear from the asymptotic set.

The basic idea is that M_2 will have the word w encoded in its counters, and a guess k that it keeps constant. It iterates through all choices of l (checking all values infinitely many times) just like in the proof of Theorem 4, and for each l, it then starts iterating through all m. When the case $(k, l, m, w) \in L$ occurs, it sends w to the left. We omit the details of outputting w at a sufficient speed. After outputting w, M_2 waits for M_1 to respond in order not to send multiple messages at once (which would add entropy to Z). The sofic rule of Z makes sure M_2 is in a waiting state if a message is on its way. When the message w_{ok} is received from M_1, M_2 continues its computation. The message w is sent infinitely many times iff it is a solution to L and the guess k was correct.

The machine M_1 behaves similarly. It waits for a message from M_2, and when one is received, it sends the message w_{ok} to the right. Note that there are only finitely many different forms of points in X (and in particular Y), so the rule of the sofic shift can share this information about w between M_1 and M_2.

Now it is clear that if the automaton is started from a good point, we obtain only words of Y in the asymptotic set. Also, for any $w \in \mathcal{B}(Y)$, a good point puts it in the asymptotic set if M_2 uses w as a message. As for bad points, we note that if only one counter machine occurs in a point, or a counter has an infinite value, then the worst that can happen is that the flow of messages stops at one point, leaving only the periodic medium – which is in Y – in the asymptotic set. □

In particular, all limit sets are asymptotic sets in the following sense:

Corollary 2. *Let X be a countable sofic shift and $f : X \to X$ a CA with limit set Y. Then there exists a countable sofic shift $Z \supset X$ and a CA $g : Z \to Z$ such that the asymptotic set of g is Y.*

Proposition 3 shows that Z above cannot be chosen equal to X in general.

7 Conclusions and Future Work

In this paper, we have studied cellular automata on sofic shifts mainly from the computational point of view. We proved a useful result about the long-term evolution of points under such automata (the Starfleet Lemma), and showed the decidability of some dynamical properties (nilpotency and periodicity) and undecidability of others (stability). It seems that those properties that depend on the exact behavior of spaceships are likely to be decidable. The Starfleet Lemma is, in these cases, a very powerful tool. We also studied the computational capacity and complexity of asymptotic sets, which turned out to be quite high.

We have already proved several results about the dynamics of cellular automata on countable sofic shifts which did not make it into this article due to lack of space. Future work might involve studying the relations between dynamical systems defined in this way. In particular, the conjugacy problem of countable sofic shifts, which seems more combinatorial than the general conjugacy problem, but challenging nevertheless, is still unsolved.

Acknowledgements. We'd like to thank Jarkko Kari for pointing out that Lemma 3 holds even in the case of a general subshift, Pierre Guillon for his useful comments, in particular on the last section, and the anonymous referees for their suggestions on the overall presentation and structure of the article.

References

1. Guillon, P., Richard, G.: Asymptotic behavior of dynamical systems and cellular automata. ArXiv e-prints (2010)
2. Kari, J.: The nilpotency problem of one-dimensional cellular automata. SIAM J. Comput. 21, 571–586 (1992)
3. Kari, J.: Rice's theorem for the limit sets of cellular automata. Theoret. Comput. Sci. 127, 229–254 (1994)
4. Ballier, A., Durand, B., Jeandel, E.: Structural aspects of tilings. In: Susanne Albers, P.W. (ed.) Proceedings of the 25th Annual Symposium on the Theoretical Aspects of Computer Science, Bordeaux, France, pp. 61–72, 11 pages. IBFI Schloss Dagstuhl (2008)
5. Morita, K.: Universality of a reversible two-counter machine. Theoretical Computer Science (1996)
6. Guillon, P., Richard, G.: Asymptotic behavior of dynamical systems and cellular automata. ArXiv e-prints (April 2010)

Computing Lempel-Ziv Factorization Online

Tatiana Starikovskaya

Lomonosov Moscow State University, Moscow, Russia
tat.starikovskaya@gmail.com

Abstract. We present an algorithm which computes the Lempel-Ziv factorization of a word W of length n on an alphabet Σ of size σ online in the following sense: it reads W starting from the left, and, after reading each $r = O(\log_\sigma n)$ characters of W, updates the Lempel-Ziv factorization. The algorithm requires $O(n \log \sigma)$ bits of space and $O(n \log^2 n)$ time. The basis of the algorithm is a sparse suffix tree combined with wavelet trees.

1 Introduction

The Lempel-Ziv factorization (further LZ-factorization for short) of a word W is a decomposition $W = f_1 f_2 \dots f_z$, where a factor f_i, $1 \leq i \leq z$, is either a character that does not occur in $f_1 f_2 \dots f_{i-1}$ or the longest prefix of $f_i \dots f_z$ that occurs in $f_1 f_2 \dots f_i$ at least twice [1, 2].

The most famous application of the LZ-factorization is data compression (e.g. the LZ-factorization is used in gzip, WinZip, and PKZIP). Moreover, it is a basis of several algorithms [3, 4] and text indexes [5].

Let W be a word of length n on an alphabet Σ of size σ. There are many algorithms that compute the LZ-factorization in $O(n \log n)$ bits of space[1]. These algorithms use suffix trees [6], suffix automata [1] or suffix arrays [7–12] as a basis.

However, only two algorithm have been known which use $O(n \log \sigma)$ bits of space [12, 13]. The algorithms exploit similar ideas (both are based on an FM-index and a compressed suffix array). The algorithm [12] is offline and requires linear time.

The algorithm [13] is online. To understand the idea behind it, consider the factors $f_1, f_2, \dots, f_i$ of the LZ-factorization of a word X. The LZ-factorization of a word Xa, where a is a character, contains either i or $i+1$ factors: in the first case the factors are $f_1, f_2, \dots, f_{i-1}, f'_i$, where the last factor $f'_i = f_i a$; and in the second case the factors are $f_1, f_2, \dots, f_i, f_{i+1}$, where $f_{i+1} = a$. The algorithm [13] reads W and after reading each new character updates the LZ-factorization, i.e. either increases the length of the last factor by one or adds a new factor. The running time of this algorithm is $O(n \log^3 n)$.

For many practical applications dealing with large volumes of data it would be natural to allow updating the LZ-factorization only each $r > 1$ new characters

[1] In this paper log stands for $\log_2$.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 789–799, 2012.

of W, for some small parameter r, in order to reduce the running time. Unfortunately, naive application of this idea to the algorithm [13] does not improve its running time.

Here we propose a new online and linear-space algorithm which achieves a reasonable trade-off between frequency of updates and running time. The algorithm updates the LZ-factorization of W each $r = \frac{\log_\sigma n}{4}$ characters of W, requiring $O(n \log^2 n)$ time and $O(n \log \sigma)$ bits of space. It is assumed that both σ and n are known beforehand and $n \geq \sigma$. The basis of the algorithm is a sparse suffix tree combined with wavelet trees.

Let X be a word of length $|X|$ on Σ. Throughout the paper, positions in X are numbered from 1. The subword of X from position i to position j (inclusively) is denoted by $X[i..j]$. If $j = |X|$, then we write $X[i..]$ instead of $X[i..|X|]$. A word $X[i..]$ is called a suffix of X and a word $X[1..j]$ is called a prefix of X.

With each word Y of length r on Σ we associate a meta-character Y' formed by concatenating bit-representations of characters of Y. Note that a bit representation of any character of Y can be obtained from the bit representation of Y' by two shift operations. Also, Y' can be obtained from Y in $O(r)$ time by standard bit-vector operations.

2 Algorithm

Let $f_1, f_2, \dots f_z$ be the factors of the LZ-factorization of W. For the sake of clarity we describe not how to update the LZ-factorization after reading each block of letters but rather how to compute $f_1, f_2, \dots f_z$ sequentially. However, it will be easy to see that the presented algorithm can be easily modified to solve the problem we formulated in the introduction.

Suppose that $f_1, f_2, \dots f_{i-1}$ of total length ℓ_i have been computed. The algorithm consists of two procedures. The procedure $P_{<r}$ checks if $|f_i|$ is less than r and, if it is, computes f_i (Section 2.2). The procedure $P_{\geq r}$ computes f_i only if it is already known that $|f_i| \geq r$ (Section 2.3). To compute f_i the algorithm runs $P_{<r}$ first and then, if necessary, runs $P_{\geq r}$.

2.1 Data Structures

The algorithm makes use of several data structures. To explain what these data structures are, we need to give a definition a compacted trie first.

Definition 1. *A trie for a set of words S is a rooted tree edges of which are labelled by letters. For each prefix P of a word of S there must exist exactly one vertex such that P is spelled out by the path from the root of the trie to this vertex, and vice versa, a word spelled by any downward path must be a prefix of one of the words of S. A compacted trie for S can be constructed from the trie by eliminating all vertices with one son, thus forming edges that spell out words rather than single letters.*

The algorithm reads W by blocks of r letters starting from the left. After reading the t-th block of W, the first data structure is a compacted trie on suffixes of words $W[rj+1..r(j+2)]$, $j = 0..t-2$. Each explicit vertex v of the trie stores a starting position of one of the suffixes ending in the subtree rooted at v.

Let W' be the meta-word formed by replacing each block of characters of W with the corresponding meta-character. The second data structure is an implicit suffix tree for $W'[1..t]$, i.e. a compacted trie for the set of suffixes of $W'[1..t]$. This tree is also called a sparse suffix tree for $W[1..tr]$ [14–16], though the original definition of a sparse suffix tree is slightly different [17].

For each explicit vertex v of the suffix tree we store a compacted trie CT_v on words of length r corresponding to the first meta-characters on the edges outgoing from v.

Definition 2. *Consider a tree with labels on edges (a suffix tree or a trie). We say that a word X is* represented *by a vertex v (or that v* represents *X), if the path from the root of the tree to v is labelled by X.*

If the label of an edge (v, u) of the suffix tree begins with a meta-character Y', and Y is the corresponding word of length r, then we store a pointer to the edge (v, u) in the leaf of CT_v representing Y. The tries in vertices are used for navigation in the suffix tree (but not only for it). Clearly, given a vertex and a meta-character, it takes $O(r \log \sigma) = O(\log n)$ time to find the edge outgoing from the vertex, the label of which starts with the meta-character.

After reading the $(t+1)$-th block of W, we first compute $W'[t+1]$ in $O(r)$ time and then update the suffix tree by Ukkonen's algorithm [18].

Definition 3. *Block borders are positions of W of the form $pr+1$, where $p \in [1, \lfloor \frac{n}{r} \rfloor]$.*

Let B_v be a set of block borders corresponding to the starting positions of the suffixes represented by the leaves of the subtree rooted at an explicit vertex v. We store an additional data structure which allows, given v, a word $Y \in \Sigma^{[1,r]}$, and a block border b, to determine whether $B_v \setminus \{b\}$ contains a block border preceded by an occurrence of Y. If there are such block borders, the data structure reports one of them. The query takes $O(\log^2 n)$ time.

Details of implementation are not important to understand the algorithm and will be explained later, in Section 3.

Hereafter $\lfloor \frac{\ell_i}{r} \rfloor$ is denoted by ℓ'_i. We assume that the algorithm has read the first $\ell'_i + 1$ blocks of W before running the procedures $P_{<r}$ and $P_{\geq r}$.

2.2 Procedure $P_{<r}$

Lemma 1. *$W[\ell_i + 1..\ell_i + r]$ occurs in one of the words $W[rj+1..r(j+2)]$, $j = 0..\ell'_i - 1$, iff $|f_i| \geq r$ (see Fig. 1).*

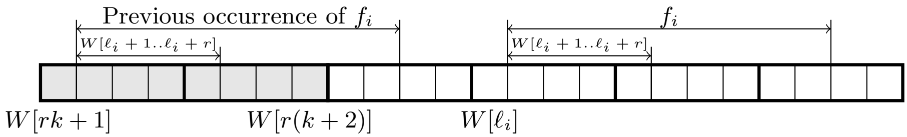


Fig. 1. Case $|f_i| \geq r$, $r = 4$. The word $W[rk+1..r(k+2)]$, $k < \ell'_i$, containing an occurrence of $W[\ell_i+1..\ell_i+r]$, is highlighted in grey. Block borders are in bold.

We traverse the trie starting at the root and following edges labelled with the characters of $W[\ell_i+1..\ell_i+r]$. Two cases are possible: either we will read out the whole word $W[\ell_i+1..\ell_i+r]$ or we will stop after reading $W[\ell_i+1..s]$, $s < \ell_i+r$, and will not be able to proceed. It follows from the lemma that in the first case $|f_i| \geq r$ and in the second case $|f_i| < r$. Moreover, it is easy to see that in the second case $|f_i|$ is equal to $|W[\ell_i+1..s]|$ and that we can report a previous occurrence of f_i in $O(1)$ time.

Obviously, $P_{<r}$ takes $O(|f_i|)$ time in both cases.

2.3 Procedure $P_{\geq r}$

$P_{\geq r}$ consists of two steps. The first one is preliminary, and during the second step we compute $|f_i|$.

The First Step. $P_{\geq r}$ starts with reading W. After reading the s-th block, it updates the data structures and checks whether $W'[\ell'_i..s]$ is represented by a leaf in the suffix tree of $W'[1..s]$. If it is, $P_{\geq r}$ proceeds to the second step. From the description of Ukkonen's algorithm [18] it follows that after the first step of $P_{\geq r}$ all suffixes starting at positions less than ℓ'_i will be represented by leaves.

Lemma 2. *During the first step at most $|f_i| + r$ characters of W will be read.*

Proof. Since s is the minimal position such that $W'[\ell'_i..s]$ is represented by a leaf, $W'[\ell'_i..s-1]$ is represented by an inner vertex in the suffix tree of $W'[1..s-1]$ and, consequently, occurs before the position ℓ'_i in W'. Therefore, $W[\ell_i+1..(s-1)r]$ occurs before the position ℓ_i (see Fig. 2) and $|f_i| \geq |W[\ell_i+1..(s-1)r]|$. The statement of the lemma easily follows.

We initialize M with $|W[\ell_i+1..(s-1)r]|$. The lemma guarantees that $|f_i| \geq M$. During the computation process we will increase M until, finally, it will become equal to $|f_i|$.

Definition 4. Depth *of a vertex v of the suffix tree is the length of the word represented by v.*

Lemma 3. *Let v be an explicit inner vertex of the suffix tree of $W'[1..s]$ with depth at least $\lfloor \frac{M}{r} \rfloor$. If a block border belongs to the set B_v, then it is not bigger than $(\ell'_i+1)r+1$.*

Proof. Indeed, a subtree rooted at v can only contain leaves representing suffixes of length at least $\lfloor \frac{M}{r} \rfloor = s - \ell'_i$, and all such suffixes start at positions $\leq \ell'_i + 1$. The statement immediately follows.

We will read a new block of characters of W and update the data structures only when $\ell_i + M$ is bigger than the position of the last read character. This will guarantee that a statement similar to the statement of the lemma will be true throughout $P_{\geq r}$.

Idea of the Second Step. Consider the first block border which intersects a previous occurrence of f_i (see Fig. 3). It divides the occurrence into two parts: the first short part equal to $W[\ell_i + 1..\ell_i + m - 1]$ and the second part equal to a prefix of $W[\ell_i + m..]$, $m \in [1, r]$.

Let f_i^m be the longest prefix of $W[\ell_i + m..]$ with at least one occurrence at a block border which is less than ℓ_i and preceded by an occurrence of $W[\ell_i + 1..\ell_i + m - 1]$. Obviously, $|f_i| = \max_{m \in [1,r]}(|f_i^m| + m - 1)$.

For each $m = 1..r$ the procedure $P_{\geq r}$ either computes $|f_i^m|$ and updates M or proves that $|f_i^m| + m - 1 \leq M$ and starts computation of $|f_i^{m+1}|$.

If $(\ell'_i + 1)r + 1 - m \leq \ell_i$, then the second step of $P_{\geq r}$ starts with computing the length q of the longest common prefix of $W[\ell_i + 1..]$ and $W[(\ell'_i + 1)r + 1 - m..]$. In order to compute q the procedure compare $W[\ell_i + 1..]$ and $W[(\ell'_i + 1)r + 1 - m..]$ character by character. If $q > M$, the procedure puts M equal to q. Then the procedure starts to work with the suffix tree.

Let $W'_{\ell_i + m}$ be a meta-word formed by grouping every r characters of $W[\ell_i + m..]$ into a single meta-character. Note that each character of $W'_{\ell_i + m}$ can be obtained by at most two shift operations from appropriate characters of W', therefore there is no need to compute $W'_{\ell_i + m}$ in advance or to store it explicitly.

Note that if a path from the root of the suffix tree to a vertex v is labelled by $W'_{\ell_i + m}[1..p]$, then $W[\ell_i + m..\ell_i + m + pr]$ occurs at all block borders of B_v, and vice versa.

This simple observation gives us the idea of how $|f_i^m|$ can be computed. We traverse the suffix tree starting at the root and following the edges labelled with the characters of $W'_{\ell_i + m}$. Let v be an explicit vertex representing a word

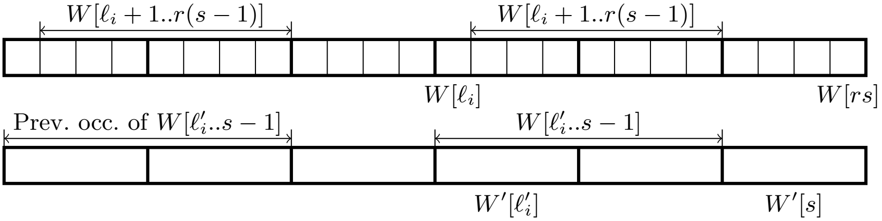


Fig. 2. Relation between $W'[\ell'_i..s-1]$ and $W[\ell_i + 1..r(s-1)]$

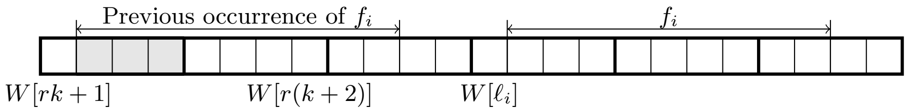


Fig. 3. A previous occurrence of f_i. The part equal to $W[\ell_i + 1..\ell_i + m - 1]$ ($m = 4$) is highlighted in grey.

$W'_{\ell_i+m}[1..p]$, where $p > \frac{M-m+1}{r}$. Clearly, $|f_i^m| \geq pr$ iff B_v contains a block border less than ℓ_i preceded by an occurrence of $W[\ell_i + 1..\ell_i + m - 1]$.

Since the depth of v is equal to $p \geq \frac{M-m+1}{r} + 1 \geq \lfloor \frac{M}{r} \rfloor$, B_v can contain only one block border bigger than ℓ_i, namely, $(\ell'_i + 1)r + 1$ (Lemma 3). So, $|f_i^m| \geq pr$ iff $B_v \setminus \{(\ell'_i + 1)r + 1\}$ contains a block border preceded by an occurrence of $W[\ell_i + 1..\ell_i + m - 1]$, and this is exactly the type of questions the additional data structure for the suffix tree can answer for (see Section 3). If there is such a border, the algorithm updates M by $pr + m - 1$ and proceeds. If not, the algorithm starts computation of $|f_i^{m+1}|$.

We have omitted several technical details from the description of the second step of $P_{\geq r}$ and give them in Appendix. We also show there that the procedure $P_{\geq r}$ takes $O(|f_i| \log^2 n + r \log^2 n)$ time.

3 Data Structures

As we have already said, our algorithm maintains two data structures. In this section we give the details and describe update procedures.

3.1 Trie

After reading $W[1..tr]$ the trie contains suffixes of words $W[rj + 1..r(j + 2)]$, $j = 0..t - 2$. To update the trie after reading the $(t + 1)$-th block of characters we first check if $W[r(t - 1) + 1..r(t + 1)]$ is represented in the trie. To do that we traverse the trie starting at the root and following edges labelled with the characters of $W[r(t - 1) + 1..r(t + 1)]$. If we read out the whole word, then $W[r(t-1)+1..r(t+1)]$, and, consequently, all its suffixes are represented in the trie. If not, we add all suffixes of $W[r(t - 1) + 1..r(t + 1)]$, including the word itself, to the trie.

Lemma 4. *The trie occupies $o(n)$ bits and its maintenance takes $O(n \log \sigma)$ time.*

Proof. Due to our choice of r, there are at most $\sigma^{2r} = \sigma^{\frac{\log_\sigma n}{2}} = n^{\frac{1}{2}}$ different words of length $2r$ on Σ. Therefore, the trie has $o(n^{\frac{1}{2}} r) = o(n)$ leaves and, consequently, edges, since each inner vertex of the trie has at least two sons. All labels are subwords of W, and we can specify them by their starting and final positions. So, the trie occupies $o(n)$ space.

To check if the words $W[rj+1..r(j+2)]$, $j = 0..\frac{n}{r} - 2$, are represented in the trie one needs $O(n \log \sigma)$ time in total. During the algorithm we add suffixes of at most $n^{\frac{1}{2}} < \frac{n}{r^2}$ words. All suffixes of a word of length $2r$ can be added to the trie in $O(r^2 \log \sigma)$ time, so we get the announced time bound.

Finally, suppose that we create a new vertex v in the process of adding a suffix $W[p..r(k+2)]$ of the word $W[rk+1..r(k+2)]$ to the trie. Then we just remember the position p as a starting position of a suffix ending in the subtree rooted at v. This completes the description of the update procedure.

3.2 Suffix Tree

The suffix tree is updated by Ukkonen's algorithm [18]. When we create a new edge outgoing from a vertex v with the first character of the label equal to $W'[k]$, we add $W[(k-1)r+1..kr]$ to CT_v.

Below we describe additional data structures which allow, given an explicit vertex v, a word $Y \in \Sigma^{[1,r]}$, and a block border b, to determine whether $B_v \setminus \{b\}$ contains a block border preceded by an occurrence of Y in $O(\log^2 n)$ time.

We define a meta-character c_{min} as follows: reverse the bit representation of Y and then append $(r - |Y|) \log \sigma$ zeros to it. A meta-character c_{max} is defined in a similar way, but ones are appended instead of zeros. Obviously, a block border $pr+1$ is preceded by an occurrence of Y iff the reverse of the bit representation of $W'[p-1]$ lies in the interval $[c_{min}, c_{max}]$.

Let p_i be the starting position of the suffix represented by the i-th leaf in the left-to-right order on the leaves of the suffix tree. Consider virtual sequences $GBWT$, $GBWT[i]$ equal to the reverse of the bit representation of $W'[p_i - 1]$, and B, $B[i]$ equal to the block border $p_i r + 1$. We store $GBWT$ and B using dynamic wavelet trees (Theorem 9 [19]). In Theorem 9 [19] σ denotes the size of the alphabet of the sequence, i.e. $\log \sigma = O(\log n)$ for $GBWT$ and B, and we put $q = 2$. In this case, the dynamic wavelet trees occupy $O(n \log \sigma)$ bits of space and allow to read any element and to add a new element after the i-th one in $O(\log^2 n)$ time.

Let $l(v)$ and $r(v)$ be the minimal and the maximal ranks of leaves of the subtree rooted at v in the left-to-right order on the leaves of the suffix tree. Then $B_v = \{B[k] | k \in [l(v), r(v)]\}$ and the subset $B_v^Y \subseteq B_v$ of block borders preceded by Y can be defined as $B_v^Y = \{B[k] | k \in [l(v), r(v)]$ and $GBWT[k] \in [c_{min}, c_{max}]\}$. Clearly, $B_v \setminus \{b\}$ contains a block border preceded by an occurrence of Y iff B_v^Y contains a block border different from b.

Each block border belonging to B_v^Y can be retrieved in $O(\log^2 n)$ time (see also [20]). Obviously, it is enough to retrieve at most two block borders to determine whether B_v^Y contains a block border different from b.

It remains to show how the minimal and the maximal ranks of the leaves in the subtree rooted at v can be computed. The data structure we will use is similar to the one from [21].

We maintain a dynamic doubly-linked list EL corresponding to the Euler tour of the current suffix tree. Each internal vertex of the suffix tree is stored in two copies in EL, corresponding respectively to the first and last visits of the vertex during the Euler tour. Leaves of the suffix tree are kept in one copy. Observe that the leaves of the suffix tree appear in EL in the "left-to-right" order, although not consecutively.

We also maintain a balanced binary tree, denoted BT, whose leaves are elements of EL. Note that the size of BT is bounded by $2\frac{n}{r}$ and the height of BT is $O(\log n)$. Since the leaves of the suffix tree are a subset of the leaves of BT, we call them *suffix leaves* to avoid the ambiguity. Each internal vertex u of BT stores the number of the suffix leaves in the subtree of BT rooted at u.

The minimal rank of a leaf in the subtree rooted at v is the number of the suffix leaves in EL preceding the first copy of v in EL plus one. This number can be computed in $O(\log n)$ time by following the path from the leaf of BT corresponding to this copy to the root of BT and summing up the number of the suffix leaves in the subtrees rooted at the left sons of the vertices on the path. The maximal rank can be computed in a similar way.

Now we should explain how to update the additional data structures. When a new vertex v is added to a suffix tree, the following updates should be done (in order):

(i) insert v at the right place of the list EL (in two copies if v is an internal vertex),
(ii) rebalance the tree BT if needed,
(iii) if v is a leaf of the suffix tree (i.e. a suffix leaf of BT), update information about the number of suffix leaves in BT.

To see how update (i) works, we have to recall how suffix tree is updated when a new document is inserted. Two possible updates are creation of a new internal vertex v by splitting an edge into two (edge subdivision) and creating a new leaf u as a child of an existing vertex. In the first case, we insert the first copy of v right after the first copy of its parent, and the second copy right before the second copy of its parent. In the second case, the parent of u has already at least one child, and we insert u either right after the second (or the only) copy of its left sibling, or right before the first (or the only) copy of its right sibling.

Rebalancing the tree BT (update (ii)) is done using standard methods. Observe that during the rebalancing we may have to adjust the information about the number of the suffix leaves for internal vertices, but this is easy to do as only a constant number of local modifications is done at each level.

Update (iii) is triggered when a new leaf u is created in the suffix tree and added to EL. We then have to follow the path in BT from the new leaf u to the root and possibly update the information about the number of suffix leaves for all vertices on this path. These updates are straightforward. All these steps take $O(\log n)$ time.

To update the wavelet trees after adding a new leaf to the suffix tree, we only need to know the rank of this leaf in the left-to-right order on the leaves of the suffix tree, and as we have already explained it can be computed in $O(\log n)$ time using BT.

Lemma 5. *The suffix tree and additional data structures occupy $O(n \log \sigma)$ bits and their maintenance takes $O(n \log^2 n)$ time.*

Proof. The suffix tree has at most $\frac{n}{r}$ leaves and therefore $O(\frac{n}{r})$ edges. We specify labels of edges by their starting and final positions in W'. So, the suffix tree occupies $O(n \log \sigma)$ bits.

Tries in vertices of the suffix tree have $O(\frac{n}{r})$ leaves in total and occupy $O(n \log \sigma)$ bits as well (labels are specified by their starting and final positions in W). Finally, BT, EL and dynamic wavelet tree use $O(\frac{n}{r} \log n) = O(n \log \sigma)$ bits of space.

Ukkonen's algorithm [18] takes $O(\frac{n}{r} \log n)$ time (additional $\log n$ appears because of the cost of navigation). To update tries in the vertices of the suffix tree we need $O(\frac{n}{r} \log n) = O(n \log \sigma)$ time. All wavelet tree updates take $O(\frac{n}{r} \log^2 n) = O(n \log n \log \sigma) = O(n \log^2 n)$ time. And finally, update of BT and EL takes $O(\frac{n}{r} \log n) = O(n \log \sigma)$ time.

4 Results and Conclusions

To conclude, we prove the following theorem.

Theorem 1. *The presented algorithm computes the Lempel-Ziv factorization of a word W in $O(n \log^2 n)$ time and $O(n \log \sigma)$ bits of space.*

Proof. Lemmas 4 and 5 guarantee that the data structures occupy $O(n \log \sigma)$ bits of space in total and that their maintenance takes $O(n \log^2 n)$ time.

To compute f_i, first $P_{<r}$ is run. As we have proved, it takes $O(|f_i|)$ time. $P_{\geq r}$ is run only when $|f_i| \geq r$ (i.e., at most $\frac{n}{r}$ times) and takes $O((|f_i| + r) \log^2 n)$ time. Therefore, the total time spent by procedures $P_{<r}$ and $P_{\geq r}$ is $O(n \log^2 n)$, and this completes the proof.

It is easy to see that the described algorithm can be implemented as online algorithm with the same running time and working space.

Acknowledgement. The author has been supported by a grant 10-01-93109-CNRS-a of the Russian Foundation for Basic Research and by the mobility grant funded by the French Ministry of Foreign Affairs through the EGIDE agency.

The author thank Simon J. Puglisi, who proposed to consider the problem of computation of the Lempel-Ziv factorization on a sparse suffix tree, and Gregory Kucherov and Alexei Lvovich Semenov for very helpful discussions.

References

1. Crochemore, M.: Transducers and repetitions. Theor. Comput. Sci. 45, 63–86 (1986)
2. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23(3), 337–343 (1977)

3. Kolpakov, R., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: Proceedings of the 1999 Symposium on Foundations of Computer Science, pp. 596–604. IEEE Computer Society (1999)
4. Gusfield, D., Stoye, J.: Linear time algorithms for finding and representing all the tandem repeats in a string. J. Comput. Syst. Sci. 69, 525–546 (2004)
5. Kreft, S., Navarro, G.: Self-indexing Based on LZ77. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 41–54. Springer, Heidelberg (2011)
6. Rodeh, M., Pratt, V.R., Even, S.: Linear algorithm for data compression via string matching. J. ACM 28, 16–24 (1981)
7. Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. J. of Discrete Algorithms 2, 53–86 (2004)
8. Chen, G., Puglisi, S.J., Smyth, W.F.: Lempel-Ziv factorization using less time & space. Mathematics in Computer Science 1(4), 605–623 (2008)
9. Crochemore, M., Ilie, L.: Computing longest previous factor in linear time and applications. Inf. Process. Lett. 106, 75–80 (2008)
10. Crochemore, M., Ilie, L., Iliopoulos, C.S., Kubica, M., Rytter, W., Waleń, T.: LPF Computation Revisited. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 158–169. Springer, Heidelberg (2009)
11. Crochemore, M., Ilie, L., Smyth, W.F.: A simple algorithm for computing the Lempel-Ziv factorization. In: Proceedings of the Data Compression Conference, pp. 482–488. IEEE Computer Society, Washington, DC (2008)
12. Ohlebusch, E., Gog, S.: Lempel-Ziv Factorization Revisited. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 15–26. Springer, Heidelberg (2011)
13. Okanohara, D., Sadakane, K.: An Online Algorithm for Finding the Longest Previous Factors. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 696–707. Springer, Heidelberg (2008)
14. Chiu, S.-Y., Hon, W.-K., Shah, R., Vitter, J.S.: Geometric Burrows-Wheeler transform: Linking range searching and text indexing. In: Proceedings of the Data Compression Conference, pp. 252–261. IEEE Computer Society (2008)
15. Hon, W.-K., Shah, R., Thankachan, S.V., Vitter, J.S.: On Entropy-Compressed Text Indexing in External Memory. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 75–89. Springer, Heidelberg (2009)
16. Chiu, S.-Y., Hon, W.-K., Shah, R., Vitter, J.S.: I/O-efficient compressed text indexes: From theory to practice. In: Proceedings of the Data Compression Conference, pp. 426–434. IEEE Computer Society (2010)
17. Kärkkäinen, J., Ukkonen, E.: Sparse Suffix Trees. In: Cai, J.-Y., Wong, C.K. (eds.) COCOON 1996. LNCS, vol. 1090, pp. 219–230. Springer, Heidelberg (1996)
18. Ukkonen, E.: On-line construction of suffix trees. Algorithmica, 249–260 (1995)
19. Mäkinen, V., Navarro, G.: Dynamic entropy-compressed sequences and full-text indexes. ACM Trans. Algorithms 4, 32:1–32:38 (2008)
20. Mäkinen, V., Navarro, G.: Position-Restricted Substring Searching. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 703–714. Springer, Heidelberg (2006)
21. Kucherov, G., Nekrich, Y., Starikovskaya, T.: Cross-Document Pattern Matching. In: Kärkkäinen, J., Stoye, J. (eds.) CPM 2012. LNCS, vol. 7354, pp. 196–207. Springer, Heidelberg (2012)

Appendix: Technical Details of the Second Step of $P_{\geq r}$

Here we explain technical details of the second step of the procedure $P_{\geq r}$ and show that $P_{\geq r}$ does not take too much time.

First of all, Lemma 3 works only for inner vertices. If during the traverse we arrive to a leaf of the suffix tree, we first check if this leaf corresponds to a block border less than $\ell_i + 1$ and then check if the border is preceded by an occurrence of $W[\ell_i + 1..\ell_i + m - 1]$ using a character-by-character comparison. After that, the algorithm proceeds as described earlier.

Secondly, suppose that during the traverse we stop in a vertex v representing a word $W'_{\ell_i+m}[1..p]$ and cannot proceed. It follows that $W[\ell_i + m..(p+1)r]$ does not occur at block borders of $W[1..\ell_i]$ preceded by an occurrence of $W[\ell_i+1..\ell_i+m-1]$. However, this can be false for a word $W[\ell_i + m..pr + q]$, $q < r$. Next two paragraphs explain how to find the biggest such q.

Two cases are possible depending on whether v is implicit or explicit. Let v be implicit and u be the lower end of the edge containing v. To find q, we first ask if $B_u \setminus \{(\ell'_i + 1)r + 1\}$ contains a block border preceded by an occurrence of $W[\ell_i + 1..\ell_i + m - 1]$. If it does, we compare the word corresponding to the next meta-character on the edge with the word corresponding to the next meta-character of W'_{ℓ_i+m} character by character to find the length of their longest common prefix, which obviously will be equal to q.

Suppose now that v is an explicit vertex. We traverse CT_v starting at the root and following the edges labelled with the characters of $W[pr + 1..(p+1)r]$. Let u be an explicit vertex of CT_v representing a word $W[pr + 1..pr + t]$, where $pr+t+m-1 > M$. Suppose that $u_1, u_2, \ldots, u_k$ are the sons of v corresponding to the leaves of the subtree of CT_v rooted at u. Obviously, a word $W[\ell_i + m..pr + t]$ occurs at all block borders of $B_{u_1} \cup B_{u_2} \cup \ldots \cup B_{u_k}$. Moreover, the set $B_{u_1} \cup B_{u_2} \cup \ldots \cup B_{u_k}$ can contain only one block border bigger than ℓ_i, namely, $(\ell'_i + 1)r + 1$ (Lemma 3).

In each vertex u with such properties we ask whether the set $B_{u_1} \cup B_{u_2} \cup \ldots \cup B_{u_k} \setminus \{(\ell'_i + 1)r + 1\}$ contains a block border preceded by $W[\ell_i..\ell_i + m - 1]$. From the description of the additional data structure we store for the suffix tree (Section 3) it is quite obvious that such queries also can be answered in $O(\log^2 n)$ time.

The following lemma estimates the time spent during $P_{\geq r}$, not including the time for updates of the data structures.

Lemma 6. *The procedure $P_{\geq r}$ takes $O(|f_i| \log^2 n + r \log^2 n)$ time.*

Proof. First step of $P_{\geq r}$ reads $O(|f_i| + r)$ characters of W (Lemma 2).

To follow f_i^m down in the suffix tree we need $O(\frac{|f_i^m|}{r} \log n) = O(|f_i| \log \sigma)$ time (remember that search of an appropriate edge in a vertex takes $O(\log n)$ time). Since after each query to the additional data structure we either increase M or proceed to the computation of f_i^{m+1}, there are at most $r + |f_i|$ queries. A query takes $O(\log^2 n)$ time (see Section 3). Therefore, the total time spent during the second step of $P_{\geq r}$ is $O((r + |f_i|) \log^2 n + r \log \sigma |f_i|) = O(|f_i| \log^2 n + r \log^2 n)$.

On Two Stronger Versions of Dejean's Conjecture

Igor N. Tunev and Arseny M. Shur

Ural Federal University, Ekaterinburg, Russia
itnvi@mail.ru, arseny.shur@usu.ru

Abstract. Repetition threshold is the smallest number $\mathsf{RT}(n)$ such that infinitely many n-ary words contain no repetition of order greater than $\mathsf{RT}(n)$. These "extremal" repetition-free words are called threshold words. All values of $\mathsf{RT}(n)$ are now known, since the celebrated Dejean's conjecture (1972) was finally settled in 2009. We study two questions about threshold words. First, does the number of n-ary threshold words grow exponentially with length? This is the case for $3 \le n \le 10$, and a folklore conjecture suggests an affirmative answer for all $n \ge 3$. Second, are there infinitely many n-ary threshold words containing only finitely many different repetitions of order $\mathsf{RT}(n)$? The answer is "yes" for $n = 3$, but nothing was previously known about bigger alphabets.

For odd $n = 7, 9, \dots, 101$, we prove the strongest possible result in this direction. Namely, there are *exponentially many* n-ary threshold words containing *no repetitions of order* $\mathsf{RT}(n)$ *except for the repeats of just one letter.*

1 Introduction

Families of combinatorial objects parametrized by a certain numerical parameter appear in many areas of mathematics and computer science. Many of them have "phase transitions", when the properties of objects sharply change at some threshold value of the parameter. Evaluating threshold values and studying the properties of "threshold" objects is an important and challenging task connected to deep properties of the objects in question. In this paper we study repetition-free words. Over any fixed alphabet, these natural and well-known objects form a family parametrized by the maximum order of "admitted" repetitions.

Recall that the *exponent* of a word w is the ratio between its length and its minimal period: $\mathsf{exp}(w) = |w|/\mathsf{per}(w)$. If this ratio equals $\beta > 1$, then w is a *fractional power* (β*-power*). Fractional powers constitute a natural and important class of repetitions in words. A word is called β*-free* for some rational $\beta > 1$ if all its factors have exponents less than β. The *repetition threshold* function $\mathsf{RT}(n)$ was introduced by Dejean [6], see also [2]. Dejean's conjecture, now fully resolved, see [4–6,12], states that the set of n-ary β-free words is infinite if and only if $\beta > \mathsf{RT}(n)$, where

$$\mathsf{RT}(3) = 7/4,\ \mathsf{RT}(4) = 7/5,\ \text{and } \mathsf{RT}(n) = n/(n-1) \text{ otherwise.}$$

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 800–812, 2012.

The n-ary words that are β-free for any $\beta > \mathsf{RT}(n)$ are called *threshold words* and constitute the *threshold language* T_n.

Dejean's conjecture boosted the interest to the structure and properties of threshold languages. We mention two groups of questions concerning these languages. The first group consists of questions about the growth properties of threshold languages. It is known that T_2 grows polynomially with length [13], while the folklore Exponential conjecture says that T_n grows exponentially for any $n \geq 3$. This conjecture was settled for $n \leq 10$ by Ochem, Kolpakov, and Rao [7,9]. Exponential conjecture was strengthened in [15] to the Growth Rate conjecture: T_n grows exponentially at base that tends to a limit $\alpha \approx 1.242$ as n approaches infinity.

If the Growth Rate conjecture holds, there is enough room for different additional restrictions on threshold words. The second group contains the questions of the form "does the set of threshold words remain infinite under some restriction?". Such restrictions may concern, e.g., the densities of letters, see [10,12], or some of the repetitions of smaller order. The latter type of restriction was introduced by Shallit [14] who considered binary words simultaneously avoiding some avoidable power and also all but finitely many squares (i.e., repetitions of order $\mathsf{RT}(2)$). Badkobeh and Crochemore [1] proved that there are infinitely many ternary threshold words containing only two $\mathsf{RT}(3)$-powers, and conjectered a similar result for quaternary words. As for bigger alphabets, nothing was known about this type of restrictions up to now.

The aim of this paper is to prove the following theorem, linking up the two groups of questions mentioned above. Let $n \geq 5$. It is easy to see that any threshold word over n letters contains the factors of the form $a_1a_2\cdots a_{n-1}a_1$ which we call *trivial* $\mathsf{RT}(n)$*-powers*. A threshold word is said to be *pure* if it contains only trivial $\mathsf{RT}(n)$-powers.

Theorem 1. *For any odd n, $7 \leq n \leq 101$, the set of n-ary pure threshold words is infinite and, moreover, has exponential growth.*

Theorem 1 settles the Exponential conjecture for the listed alphabets and shows that all non-trivial "threshold" repetitions can be simultaneously avoided by threshold words. So, threshold languages are big indeed in spite of the fact that the proof of their infiniteness took 37 years. Also, our proof can be considered as an alternative proof of Dejean's conjecture for the listed alphabets. Finally, it seems very probable that the statement of Theorem 1 can be extended for all alphabets with at least 5 letters; this is our *Conjecture 1.*

The text is organized as follows. A new method of constructing threshold words over the alphabets with at least 5 letters is described in Sect 2. A particular implementation of this method given in Sect. 3 proves Theorem 1. Note that the construction is given explicitly, but its validation requires some computer checks based on standard pattern matching and search algorithms.

2 Uniform Sets

In this section, we show that Dejean's conjecture and Exponential conjecture can be confirmed for a given alphabet by constructing a special finite set of threshold words. We exhibit "weak" and "strong" version of such sets. Strong version uses only pure threshold words.

2.1 Preliminaries and Auxiliary Results

We study words over finite alphabets and use standard notions of length, concatenation, factor, prefix, suffix, power, period, exponent, etc, see, e.g., [8]. We write Σ_n for an unspecified n-letter alphabet, and * for the Kleene star operation (thus, Σ_n^* is the set of all n-ary words, including the empty word λ). The notation w^ω stands for the ω-word (i.e., right-infinite word) obtained by concatenating infinitely many copies of w. We say that words u and w are *equivalent* and write $u \sim w$ if w can be obtained from u by some permutation of the alphabet. Words u and w are *conjugates* if $u = xy$, $w = yx$ for some words x, y. Any factor of w can be written as $w[i...j]$ for some i, j; $w[1...|w|] = w$. A suffix of an ω-word can be written as $w[i...]$.

A word w is a β-power, $1 < \beta < 2$, if $w = xyx$ such that $|w|/|xy| = \beta$ and $\mathsf{per}(w) = |xy|$ is the minimal period of w. The number $\mathsf{ex}(w) = |x|$ is the *excess* of w. A word w is *unbordered* if it cannot be represented as $w = xyx$ for a non-empty word x. Recall that n-ary threshold words are exactly the words containing no β-powers for all β greater than the repetition threshold $\mathsf{RT}(n)$.

Languages are just the subsets of some Σ_n^*. A language is *factorial* if it contains all factors of every its element, and is *symmetric* if it contains all words equivalent to any of its elements. *Combinatorial complexity* of a language $L \subseteq \Sigma_n^*$ is the function $\mathsf{C}_L(l) = |L \cap \Sigma^l|$. A language is *exponential* if its combinatorial complexity has exponential growth.

A *substitution* over the alphabets Σ_k, Σ_n is any function $h : 2^{\Sigma_k^*} \to 2^{\Sigma_n^*}$ defined by its action on letters as follows:

$$h(L) = \bigcup_{w \in L} h(w), \quad h(a_1 \cdots a_n) = h(a_1) \cdots h(a_n), \quad h(\lambda) = \lambda.$$

If the image of any letter consists of a single word, then h is a *morphism.* The tools similar to the following lemma were widely used to prove exponentiality of languages since [2,3].

Lemma 1. *Let $K \subset \Sigma_k^*$, $L \subset \Sigma_n^*$ be symmetric infinite factorial languages, h be a substitution over the alphabets Σ_k, Σ_n such that $h(1) = w_1, \ldots, h(k-1) = w_{k-1}$, $h(k) = \{w_k, w_k'\}$ for some words $w_1, \ldots, w_k, w_k'$ of length $m > 1$, and $h(K) \subseteq L$. Then L has exponential complexity.*

Proof. For any $u \in \Sigma_k^*$, one has $|h(u)| = 2^{|u|_k}$, where $|u|_k$ is the number of occurrences of the letter k in u. By condition, if $u \in K$, then all words from the equivalence class of u belong to K. This class can be partitioned into k subclasses of equal size, having h-images of size $2^{|u|_1}, \ldots, 2^{|u|_k}$, respectively. So, if the equivalence class of u has the cardinality M, then its image has the cardinality

$$M(2^{|u|_1} + \ldots + 2^{|u|_k})/k \geq M \cdot 2^{l/k},$$

where $l = |u|$. Since the set $K \cap \Sigma_k^l$ is a union of equivalence classes, we have

$$\mathsf{C}_L(ml) \geq |h(K \cap \Sigma_k^l)| \geq \mathsf{C}_K(l) \cdot 2^{l/k} \geq 2^{l/k}.$$

Since m and k are constants while l runs over the set of positive integers, the function C_L grows exponentially.

The function $\mathsf{lcp}(u,v)$ [$\mathsf{lcs}(u,v)$] returns the length of the longest common prefix [resp., suffix] of the words u and v. For a set V, the notation $\mathsf{lcp}(V)$, $\mathsf{lcs}(V)$ refers to the maximums of the corresponding functions over all pairs of distinct elements of V. We write $\mathsf{F}(u)$ [$\mathsf{P}(u), \mathsf{S}(u)$] for the set of all factors [resp., prefixes, suffixes] of the word u. Suppose that V is a nonempty set of words, $w \in V^*$, $v \in V$ is a factor of w. If $w = w_1 v w_2$, where $w_1, w_2 \in V^*$, then we say that v occurs as a block in w. The set V is *synchronized* if for any $v \in V$, $w \in V^*$, all occurrences of v in w are blocks.

A set $\mathsf{V} \subseteq \Sigma_n^*$ is called *ℓ-uniform* if the following conditions are satisfied:
(U1) all words in V have the length ℓ;
(U2) all words in V are unbordered;
(U3) for any distinct words $v_1, v_2 \in \mathsf{V}$, the word $v_1 v_2$ is threshold;
(U4) for any distinct words $v_1, v_2, v_3 \in \mathsf{V}$, $\mathsf{lcs}(v_1, v_2) + \mathsf{lcp}(v_2, v_3) \leq \frac{\ell}{n-1}$.

Remark 1. The condition (U3) implies $\mathsf{lcp}(v_1, v_2), \mathsf{lcs}(v_1, v_2) \leq \frac{\ell}{n-1}$, but for our purposes, a stronger condition (U4) is needed.

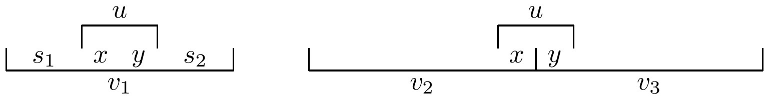
Lemma 2. *Any ℓ-uniform set is synchronized.*

Proof. If some occurrence of u in some $v \in \mathsf{V}^*$ is not a block, then by (U1) there exist $v_1, v_2 \in \mathsf{V}$ such that v_1 has a nonempty suffix x, v_2 has a nonempty prefix y, and $xy = u$. Neither of the words v_1, v_2 can be equal to u in view of (U2). Hence the word $v_1 u$ contains a square xx, which is impossible by (U3).

Lemma 3. *Let V be an ℓ-uniform set, $v_1, v_2, v_3, v_4 \in \mathsf{V}$.*
(1) *if $v_1 \neq v_2$ and $u \in \mathsf{F}(v_1) \cap \mathsf{F}(v_2)$, then $|u| \leq \frac{\ell}{n-1}$;*
(2) *if $u = xy$ such that $x, y \neq \lambda$, $u \in \mathsf{F}(v_1)$, $x \in \mathsf{S}(v_2)$, $y \in \mathsf{P}(v_3)$, then $|u| \leq \frac{\ell}{n-1}$;*
(3) *if $v_1 \neq v_3$, $v_2 \neq v_4$, and $u = xy = x'y'$ such that $x, y, x', y' \neq \lambda$, $x \in \mathsf{S}(v_1)$, $x' \in \mathsf{S}(v_3)$, $y \in \mathsf{P}(v_2)$, $y' \in \mathsf{P}(v_4)$, then $|u| \leq \frac{2\ell}{n-1}$ and $x = x'$.*

Proof. If $u \in \mathsf{F}(v_1) \cap \mathsf{F}(v_2)$, then each of the words $v_1 v_2$ and $v_2 v_1$ has a periodic factor with the excess $|u|$. The shortest of these two factors has the period at most ℓ. By (U3), $\frac{|u|}{\ell} \leq \frac{1}{n-1}$, whence (1) follows.

Let us prove (2). Put $v_1 = s_1 x y s_2$:

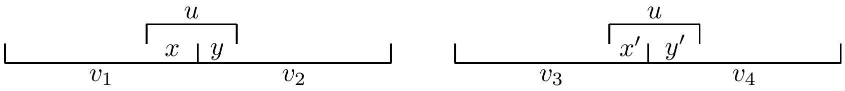


Then $x s_1 x$ is a factor of $v_2 v_1$, while $y s_2 y$ is a factor of $v_1 v_3$. If $v_2 \neq v_1 \neq v_3$, then (U3) implies $|u| = |x| + |y| \leq \frac{|s_1 x|}{n-1} + \frac{|y s_2|}{n-1} = \frac{\ell}{n-1}$.

If $v_1 = v_2 \neq v_3$, then let $s_2 = s_2'x$; the word v_2v_3 has the factor $xys_2'xy$. From (U3), we derive $|u| = |xy| \leq \frac{|xys_2'|}{n-1} < \frac{\ell}{n-1}$. The case $v_1 = v_3 \neq v_2$ is similar to the above one. Finally, if $v_1 = v_2 = v_3$, we write $v_1 = ys_1'xys_2'x$ to get

$$|u| = |x| + |y| \leq \tfrac{|ys_1'x|}{n-1} + \tfrac{|ys_2'x|}{n-1} = \tfrac{\ell}{n-1}.$$

The condition of statement (3) is illustrated by the following picture. The required inequality follows immediately from (1), (2).



If $|x| > |x'|$, as in the picture, then some nonempty suffix of x (and then, of v_1) is a prefix of y' (and of v_4). By (U2), $v_1 \neq v_4$. The word v_1v_4 contains a square in contradiction with (U3). The assumption $|x'| > |x|$ leads to a similar contradiction. Thus, $|x| = |x'|$, whence the result.

Lemma 4. *Let* V *be an* ℓ*-uniform set,* $v \in \mathsf{V}^*$, $w \in \mathsf{F}(v)$, $\mathsf{ex}(w) > \frac{\ell}{n-1}$. *Then* $\mathsf{per}(w) \equiv 0 \pmod{\ell}$.

Proof. Let $w = uzu$ such that $|u| = \mathsf{ex}(w)$. If u contains some word $v \in \mathsf{V}$, consider the leftmost occurrences of v in both u's. The distance between these occurrences of v in w equals $\mathsf{per}(w)$ and is divisible by ℓ in view of Lemma 2.

Next we assume that u is a factor of some $v \in \mathsf{V}$. Because of the length argument, u cannot occur in v twice. By Lemma 3(1), u is not a factor of any other element of V. By Lemma 3(2), u cannot appear on the border of two elements of V. Hence, the distance between the occurrences of u in w equals the distance between two blocks v, whence the result.

In the remaining case, Lemma 3(3) is applicable. Namely, u occurs only on the border of two blocks, and all these occurrences are equally placed with respect to such a border. So we again have $\mathsf{per}(w) \equiv 0 \pmod{\ell}$.

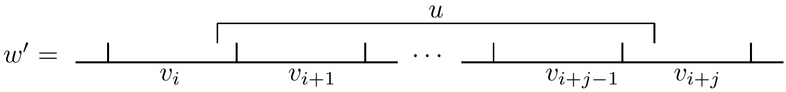
2.2 Weak Sufficient Condition for Exponential Conjecture

From now on, let $n \geq 5$. Hence, $\mathsf{RT}(n) = \frac{n}{n-1}$.

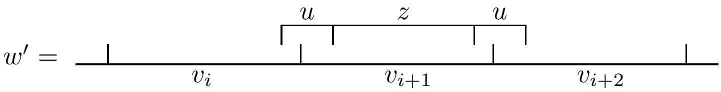
Theorem 2. *Let* $f : \Sigma_{n+k}^* \to \Sigma_n^*$ $(k \geq 1,\ n \geq 5)$ *be an injective morphism such that* $\mathsf{V} = f(\Sigma_{n+k})$ *is an* ℓ*-uniform set. Then* f *maps any threshold word over* $n{+}k$ *letters to a threshold word over* n *letters.*

Proof. Let $w = a_1 \ldots a_t \in T_{n+k}$, $w' = f(w) = v_1 \ldots v_t$, where $f(a_i) = v_i$ for all i. Furthermore, let uzu be a factor of maximum exponent of w', $\mathsf{ex}(uzu) = |u|$. Aiming at a contradiction, we assume that $\mathsf{exp}(uzu) > \frac{n}{n-1}$, i.e., $|uz| < (n{-}1)|u|$.

First consider the case $|u| > \frac{\ell}{n-1}$. By Lemma 4, $|uz| \equiv 0 \pmod{\ell}$. Due to the choice of uzu, the two occurrences of u in w' are preceded by different letters and are followed by different letters also. Then these occurrences cannot be proper factors of equal blocks. By Lemma 3(1), they cannot be proper factors of different blocks as well. Hence, there exists a factor $v_i \dots v_{i+j}$ $(i, j \geq 1)$ of w' such that the leftmost occurrence of u begins with a proper (possibly empty) suffix of v_i, contains the blocks $v_{i+1}, \dots, v_{i+j-1}$, and ends with a proper (possibly empty) prefix of v_{i+j}, see the picture below. The rightmost occurrence of u is placed in the same way inside the factor $v_{i'} \dots v_{i'+j}$. Thus, $|uz| = (i'-i)\ell$. Note that by the choice of uzu the blocks v_i and $v_{i'}$ are different, as well as the blocks v_{i+j} and $v_{i'+j}$. This means, in particular, that the mentioned suffix of v_i and the mentioned prefix of v_{i+j} both have the length at most $\frac{\ell}{n-1}$ by Lemma 3(1).



If $j = 1$, then $|u| \leq \frac{2\ell}{n-1}$. Then $|uz| < 2\ell$, whence $|uz| = \ell$. So we have $i' = i+1$, and the factor uzu is placed in w' as follows:



We see that the condition (U4) applied to the words v_i, v_{i+1}, and v_{i+2}, contradicts to the assumption $|u|{>}\frac{\ell}{n-1}$. So, from now on, $j > 1$. One has $|u| \leq (j-1)\ell + \frac{2\ell}{n-1}$ and then $|uz| < ((n-1)(j-1) + 2)\ell$. Consider the factor $w[i+1 \dots i'+j-1]$ having the excess $j-1$ and the period $i'-i = \frac{|uz|}{\ell} < (n-1)(j-1) + 2$. Since w is a threshold word, $i'-i \geq (n+k-1)(j-1)$. From the two inequalities on $i'-i$, we conclude that $k(j-1) < 2$, $k = 1$, $j = 2$, $i'-i = n$. Now consider the factor $w[i \dots i'+j] = a_i a_{i+1} \dots a_{i+n+1} a_{i+n+2}$. Since any n successive letters in an $(n+1)$-ary threshold word are distinct, the equality $a_{i+1} = a_{i+n+1}$ implies $a_i = a_{i+n+2}$. Applying (U4) to the blocks v_{i+n}, $v_{i+n+2} = v_i$, and v_{i+2}, we improve the upper bound on the length of u: $|u| \leq \ell + \frac{\ell}{n-1} = \frac{n\ell}{n-1}$. Then $|uz| < n\ell$, contradicting to the previously found equality $i' - i = n$. The case is finished.

It remains to consider the case $|u| \leq \frac{\ell}{n-1}$. We have $|uz| < \ell$ by our assumption. Hence, u does not meet the conditions of Lemma 3(3). This implies that u is a factor of some block v_i. By the restriction on $|uz|$, the whole factor uzu is contained either in $v_{i-1}v_i$ or in $v_i v_{i+1}$. Then our assumption contradicts to (U3). The theorem is proved.

Theorem 2 allows one to make the following conclusions about particular cases of Dejean's conjecture and Exponential conjecture:

Corollary 1. *Suppose that there exists an ℓ-uniform set $\mathsf{V} \subset \Sigma_n^*$ of size $n+k$.*
(1) *If Dejean's conjecture holds for $n+k$, then it holds for n.*
(2) *If Exponential conjecture holds for $n+k$, then it holds for n.*
(3) *If $k \geq 2$ and Dejean's conjecture holds for $n+k-1$, then Exponential conjecture holds for n.*

Proof. Theorem 2 implies that for any N, the number of n-ary threshold words of length ℓN is not less than the number of $(n+k)$-ary threshold words of length

N. Statements 1, 2 immediately follow from this. For the last statement, one can build a substitution of the form described in Lemma 1.

2.3 Strong Sufficient Condition for Exponential Conjecture

In order to prove the cases of Dejean's conjecture and Exponential conjecture without assumptions concerning bigger alphabets, we need a stronger version of ℓ-uniform sets. Let $\varepsilon \geq 0$. A threshold word $w \in \Sigma_n^*$ is said to be ε-*threshold* if any factor xyx of w such that $|x| \geq 3$ satisfies the inequality

$$\frac{|x|+\varepsilon}{|xy|} \leq \frac{1}{n-1} \text{ or, equivalently, } \mathsf{exp}(xyx) \leq \frac{n}{n-1} - \frac{\varepsilon}{\mathsf{per}(xyx)}. \qquad (1)$$

It can be directly verified that for w, any factor xyx of excess 2 has the period at least $2n-1$. Thus, 0-threshold words are just threshold words, while ε-threshold words with $\varepsilon > 0$ are pure threshold words having a prescribed "reserve" in the exponent of long factors. An ℓ-uniform set $\mathsf{V} \subseteq \Sigma_n^*$ is called (ℓ, ε)-*uniform* if the following stronger versions of (U3) and (U4) are satisfied:

(U3') for any distinct words $v_1, v_2 \in \mathsf{V}$, the word v_1v_2 is ε-threshold;
(U4') for any distinct words $v_1, v_2, v_3 \in \mathsf{V}$, $\mathsf{lcs}(v_1, v_2) + \mathsf{lcp}(v_2, v_3) + \varepsilon \leq \frac{\ell}{n-1}$.

Though there are no morphisms preserving the threshold language T_n [2], the existence of (ℓ, ε)-*uniform* sets allows us to build a morphic-like construction that preserves T_n. A k-*valued morphism* η over Σ_n is defined as follows. Take kn distinct words and assign them to letters, k words per letter; enumerate the words assigned to each letter a as $\eta_0(a), \ldots, \eta_{k-1}(a)$. Then the η-image of any word w is calculated from left to right such that the mth occurrence of a letter a is replaced by the word $\eta_{m \bmod k}(a)$. Note that a 1-valued morphism is just a usual injective morphism. A simple example illustrates the introduced notion.

Example 1. Suppose that a 2-valued morphism η over Σ_2 is given by $\eta_0(1) = 11$, $\eta_1(1) = 12$, $\eta_0(2) = 22$, $\eta_1(2) = 21$. Then $\eta^4(1) = 11\,12\,11\,22\,12\,11\,21\,22$.

Theorem 3. *Let $\eta : \Sigma_n^* \to \Sigma_n^*$ $(n \geq 5)$ be a 3-valued morphism such that the set $\mathsf{V} = \{\eta_i(a) \mid i = 0, 1, 2; a \in \Sigma_n^*\}$ is (ℓ, ε)-uniform for $\varepsilon = \frac{\mathsf{lcp(V)+lcs(V)}}{\ell-1}$. Then η maps any ε-threshold word to an ε-threshold word.*

Proof. Let $w = a_1 \ldots a_t \in \Sigma_n^*$ be an ε-threshold word, $w' = \eta(w) = v_1 \ldots v_t$, where $v_i \in \{\eta_0(a_i), \eta_1(a_i), \eta_2(a_i)\}$. We need to check that the exponents of all factors of w' satisfy (1). If such a factor is contained in a product of two or three successive blocks, then it satisfies (1) by (U3'), (U4'), because any three successive blocks in w' are different. In particular, all factors of w' with the excess at most $\frac{\ell}{n-1}$ satisfy (1).

Now let uzu be a factor of w' such that $\mathsf{ex}(uzu) = |u| > \frac{\ell}{n-1}$, and prove (1). W.l.o.g., assume that uzu is not contained in a longer factor of period $|uz|$. By Lemma 4, $|uz| \equiv 0 \pmod{\ell}$. Similarly to the proof of Theorem 2, we conclude that u is not contained inside any block, and consider two factors of w'. The factor $v_i \cdots v_{i+j}$ $(i, j \geq 1)$ is such that the leftmost occurrence of u begins with a proper (possibly empty) suffix of v_i, contains the blocks $v_{i+1}, \dots, v_{i+j-1}$, and ends with a proper (possibly empty) prefix of v_{i+j}. The factor $v_{i'} \cdots v_{i'+j}$ has the same properties with respect to the rightmost occurrence of u. Recall that $v_i \neq v_{i'}$ and $v_{i+j} \neq v_{i'+j}$ (otherwise, uzu can be extended to a longer word with the same period).

Let $j = 1$. Since the considered occurrences of u are preceded by different letters and followed by different letters, we have $|u| = \mathsf{lcp}(v_{i+1}, v_{i'+1}) + \mathsf{lcs}(v_i, v_{i'}) \le \frac{2\ell}{n-1} - 2\varepsilon$ by (U4'). Hence we immediately get (1) if $\mathsf{per}(uzu) \ge 2\ell$. In the case $\mathsf{per}(uzu) = \ell$ one has $uzu \in \mathsf{F}(v_i v_{i+1} v_{i+2})$ and (1) follows from (U4').

Now let $j > 1$. Since $v_{i+1} = v_{i'+1}$, we have $a_{i+1} = a_{i'+1}$. Moreover, since η is a 3-valued morphism, the letter a_{i+1} occurs at least two more times between the two mentioned positions. The distance between the two occurrences of the same letter in an n-ary threshold word cannot be less than $n{-}1$. Hence, $i'{-}i \ge 3n{-}3$. Recall that $\mathsf{per}(uzu) = (i'{-}i)\ell$, $|u| \le (j{-}1)\ell + \mathsf{lcp}(\mathsf{V}) + \mathsf{lcs}(\mathsf{V})$. Hence, in the case $j \le 3$ the word uzu satisfies (1), because $\mathsf{lcs}(V), \mathsf{lcs}(V) \le \frac{\ell}{n-1}$ by Remark 1. Now let $j \ge 4$. Then, the excess of the factor $w[i{+}1...i'{+}j{-}1]$ is at least 3. Since the word w is ε-threshold, we obtain $\frac{j-1+\varepsilon}{i'-i} \le \frac{1}{n-1}$ by (1). Now we finish the proof confirming the condition (1) for uzu:

$$\frac{|u|+\varepsilon}{|uz|} \le \frac{(j-1)\ell+\mathsf{lcp}(\mathsf{V})+\mathsf{lcs}(\mathsf{V})+\varepsilon}{(i'-i)\ell} = \frac{(j-1)\ell+\varepsilon\ell}{(i'-i)\ell} = \frac{j-1+\varepsilon}{i'-i} \le \frac{1}{n-1}.$$

Corollary 2. *Let* $\mathsf{V} \subset \Sigma_n^*$ *be an* (ℓ, ε)*-uniform set, where* $\varepsilon = \frac{\mathsf{lcp}(\mathsf{V})+\mathsf{lcs}(\mathsf{V})}{\ell-1}$.
(1) *If* $|\mathsf{V}| = 3n$*, then Dejean's conjecture holds for* n*.*
(2) *If* $|\mathsf{V}| > 3n$*, then Exponential conjecture holds for* n*.*

Proof. Theorem 3 implies that for any N, the number of ε-threshold n-ary words of length ℓN is not less than the number of such words of length N. Statement 1 follows from this. Having one more word in V, one can build an analog of the substitution from Lemma 1 and use the same argument to get statement 2.

3 Construction of Uniform Sets

In this section, we present a method of building (ℓ, ε)-uniform sets of required cardinality by means of sets of their *ternary codes*. A particular implementation of this method (Sect. 3.2) proves Theorem 1. Ternary encoding of threshold words [15] is a variation of the *Pansiot encoding* [11].

3.1 Ternary Encoding and Its Properties

In n-ary threshold words, two successive occurrences of a letter are at the distance $n{-}1$, n, or $n{+}1$, and are followed by different letters. Then a threshold word $u = a_1 \cdots a_l$ can be uniquely encoded by its first n letters and a ternary codeword $w = b_1 \cdots b_{l-n}$ over the auxiliary alphabet $\Delta = \{ ↗, ↕, ↘ \}$ such that $b_i =$ ↗ [resp., ↕, ↘] if $a_{i+n} = a_{i+1}$ [resp., $a_{i+n} = a_i$, $a_{i+n} = a_{i-1}$]. The notation refers to the "cylindric representation" of threshold words, see [15].

Remark 2. In fact, three words participate in the encoding procedure: the initial segment $u' = u[1...n]$, the encoded part $u'' = u[n{+}1...l]$, and the codeword w. Any two of these words uniquely determine the third one. So, it is convenient to "factor out" the initial segment and say that u'' encodes to w [starting with u'] and w decodes to u'' [starting with u']. In this way, encoding and decoding are length-preserving functions.

It is easy to see that if $w \in \Delta^*$ encodes a threshold word, then in it ╱ is always followed by ╲, ╲ is always preceded by ╱ (we denote each such pair as ╳), and ┃ is always followed by ╳. Below we refer to the words satisfying these conditions as Δ-words. We call a Δ-word *whole* if it is of the form ╳ $\cdots$ ╳, ┃ $\cdots$ ╳, or ╳ $\cdots$ ┃. Let w be a whole word. Then w^ω is an infinite Δ-word.

Let π be an arbitrary permutation of Σ_n, i.e., an element of the symmetric group S_n. We consider π as a word of length n and denote by $\mu(\pi, w)$ the word encoded by a Δ-word w starting with π. If $w \in \Delta^*$ is a whole word, then for any $\pi \in S_n$ the word $\mu(\pi, w)$ ends with a permutation. Thus, we can identify w with a permutation mapping π to the suffix of $\mu(\pi, w)$ of length n. Let k_w be the order of this permutation (we refer to k_w as the *order of* w). Then the word $\mu(\pi, w^{k_w})$ ends with π. Hence, we get the following lemma.

Lemma 5. *For any whole word $w \in \Delta^*$ and any $\pi \in S_n$, one has $\mu(\pi, w^\omega) = \hat{w}^\omega$, where $\hat{w} = \mu(\pi, w^{k_w})$.*

We say that the word $\hat{w}$ from Lemma 5 is the *word generated by* w. Since the avoidance properties of such words are independent of π, we suggest by default that π is the identity permutation 1_n. We prove two more simple properties, followed by the main result of this subsection.

Lemma 6. *If two whole words are conjugates, then their orders coincide.*

Proof. If some words w and u are conjugates, then so are their powers; so, we can write $w^{k_w} = xy$ and $u^{k_w} = yx$. Since xy acts as the identity permutation, the same holds for yx.

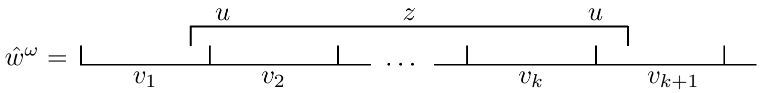
Lemma 7. *If whole words u and w are conjugates, then the word generated by u is equivalent to a conjugate of the word generated by w.*

Proof. Let $w = xy$ and $u = yx$ be whole words. Then the ω-words w^ω and u^ω are suffixes of each other. The word x, acting as a permutation, maps π to some σ. Thus, the ω-words $\mu(\pi, w^\omega)$ and $\mu(\sigma, u^\omega)$ are suffixes of each other as well. Since u and w have the same order k by Lemma 6, the words $\mu(\pi, w^k)$ and $\mu(\sigma, u^k)$ are conjugates. The words $\mu(\sigma, u^k)$ and $\mu(\pi, u^k)$ are equivalent, whence the result.

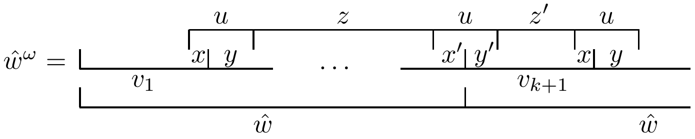
Lemma 8. *Suppose a whole word w of order $k \geq 3$ generates an ε-threshold word for some $\varepsilon \geq 0$. Then any whole word which is a conjugate of w also generates an ε-threshold word.*

Proof. In all words generated by ternary codewords, the factors of excess 1 and 2 have exponents at most $\frac{n}{n-1}$. If the statement of the lemma fails, some word generated by a (whole) conjugate of w has a factor uzu, $\mathsf{ex}(uzu) = |u| \geq 3$ such that $\mathsf{exp}(uzu) > \mathsf{exp}(s)$ for any factor s of $\hat{w}$ with excess $|u|$. W.l.o.g., $\mathsf{exp}(uzu) \geq \mathsf{exp}(x)$ for any factor x with excess $|u|$ of any word generated by a whole conjugate of w. Note that uzu is equivalent to a factor of $\hat{w}^\omega$ by Lemma 7.

Recall that $\hat{w} = v_1 \dots v_k$, where $v_i \sim v_2 \sim \dots \sim v_k$, since all these blocks are decoded from w. Moreover, for any positive integer m, the ω-word $\hat{w}^\omega[ml{+}1...]$, where $l = |v_i|$, is generated by w^ω and hence is equivalent to $\hat{w}^\omega$. In particular, any k successive blocks in $\hat{w}^\omega$ constitute a word which is equivalent to $\hat{w}$. Furthermore, we have, for any m and i, $\hat{w}^\omega[ml{+}i...] \sim \hat{w}^\omega[i...]$. So, a word equivalent to uzu begins inside v_1. W.l.o.g., we assume that this word is uzu itself. Since $|uzu| \leq lk$ and uzu is not a factor of $\hat{w}$, the word uzu must occur in $\hat{w}^\omega$ as in the following picture.



Since $v_{k+1} \dots v_{2k} = v_1 \dots v_k$, one more occurrence of u begins inside the block v_{k+1}, so we can add a few details to the previous picture:



We see that $|z'| \leq l{-}2$. On the other hand, $|z| \leq |z'|$ by the choice of uzu.

Let us write $u = xy = x'y'$, where x is the prefix of u contained in v_1 and y' is the suffix of u contained in v_{k+1} (see the picture above). Since $\hat{w}$ is a threshold word by the conditions of the lemma, we have $\mathsf{exp}(uzx'), \mathsf{exp}(yzu) \leq \frac{n}{n-1}$. Thus, $(n{-}1)|x'|, (n{-}1)|y| \leq |u| + |z|$. By simple transformations, we get

$$|x'| \leq \tfrac{|y'|+|z|}{n-2}, \quad |y| \leq \tfrac{|x|+|z|}{n-2}; \tag{2}$$

in particular, $|x'|, |y| < l$. Note that yzx' is a product of blocks. Hence, if z is a factor of a block, then yzx' is a block. But then $k = 2$, contradicting to the conditions of the lemma. Therefore, z should occur at the border of two blocks, and yzx' equals the product of these two blocks. Recall that $|z| \leq |z'| = l - |y'| - |x|$. Since $n \geq 5$, from these conditions and (2) we obtain

$$l < \tfrac{2(n-2)l-l}{n-1} = \tfrac{(n-2)(|y|+|z|+|x'|)-|z'|-|y'|-|x|}{n-1} \leq$$
$$\leq \tfrac{(n-2)|z|+|x|+|z|+|y'|+|z|-|z'|-|y'|-|x|}{n-1} \leq |z|,$$

contradicting to the condition $|z| \leq l{-}2$ obtained above. The lemma is proved.

Lemma 8 suggests a way to construct (ℓ, ε)-uniform sets: find a whole word w which generates an ε-threshold word of length ℓ over a given alphabet, and choose the other elements of the required set among the words generated by the conjugates of w. All these words have length ℓ by Lemma 6 and are ε-threshold by Lemma 8. So, the problem is to choose such an "axial" word and its conjugates in a way that ensures the fulfilment of conditions (U2), (U3'), and (U4'). Below we introduce an axial word that defines (ℓ, ε)-uniform sets over odd-size alphabets.

3.2 Axial Words for Odd-Size Alphabets

Fix an odd number $n \geq 5$. For any odd $j \geq 1$, let $d_j = (\times)^{(n+j-4)/2} \mid \times \mid$. Then one has $|d_j| = n+j$. We call the words d_j *inner blocks*. Furthermore, we define *outer blocks* $r_i = d_7 d_1 d_5 d_{2i+7} d_1 d_7$ and $r'_i = d_7 d_1 d_{2i+7} d_5 d_1 d_7$ for all numbers $i = 1, \ldots, m = (n-3)/2$. Finally, the n-ary *axial word* $w_1 = r_1 r'_1 r_1 r_2 \cdots r_m r_1 r'_1 r'_1 r'_2 \cdots r'_m$ consists of $n+1$ outer blocks. Note that $|r_i| = |r'_i| = 6n + 28 + 2i$, and then

$$|w_1| = (6n+28)(n+1) + 8 + 2(2+4+\ldots+2m) = 13n^2/2 + 32n + 75/2.$$

Now consider the following conjugates of the n-ary axial word:

- the words $w_2, \ldots, w_{n+1}$, obtained by cyclic shifts of w_1 by an integer number of outer blocks (the words $w_1, \ldots, w_{n+1}$ are referred to as "type 0" words);
- the words w_j^- and w_j^+, where $j = 1, \ldots, n+1$, obtained by a cyclic shift of the word w_j by the inner block d_7 to the left ("type –" words) and to the right ("type +" words), respectively.

Let $W = \{w_1, \ldots, w_{n+1}, w_1^-, \ldots, w_{n+1}^-, w_1^+, \ldots, w_{n+1}^+\}$. Theorem 1 follows from the next proposition and Corollaries 2(2) and 1(2) [resp., for $n \geq 9$ and $n = 7$].

Proposition 1. *Suppose that n is a fixed odd number, $7 \leq n \leq 101$, k is the order of the n-ary axial word, $\hat{w} = \mu(1_n, w^k)$ for all $w \in W$, $\hat{W} = \{\hat{w} \mid w \in W\}$, $\ell = |\hat{w}|$, and $\varepsilon = \frac{\mathsf{lcp}(\hat{W}) + \mathsf{lcs}(\hat{W})}{\ell - 1}$. Then the set $\hat{W}$ is (ℓ, ε)-uniform if $n \geq 11$, contains an (ℓ, ε)-uniform subset of cardinality 28 if $n = 9$, and an (ℓ, ε)-uniform subset of cardinality 9 if $n = 7$.*

Our *Conjecture 2* states that the set $\hat{W}$ from Proposition 1 is (ℓ, ε)-uniform for any odd $n \geq 11$. If it holds true, it implies Conjecture 1 for all mentioned alphabets.

Proof (of Proposition 1, sketched). First we calculate k with the aid of computer, getting $k \geq 4$. Next we check the condition (U4'). Note that two words from $\hat{W}$ have a common prefix or suffix of length t if and only if their codewords have a common prefix [resp., suffix] of this length. The maximum of the sum $S(w, w', w'') = \mathsf{lcp}(w, w') + \mathsf{lcp}(w', w'')$ over all triples of distinct words from W equals $S(w_3, w_{m+5}, w_{m+4}) = 24n + 116$. This is less by more than a unit than $\ell/(n-1) = k(13n/2 + 77/2 + 76/(n-1))$ if $k \geq 4$. Since $\varepsilon < 1$, we obtain (U4'). Note that (U2) follows from (U3') and the structure of $\hat{W}$. Indeed, if $\hat{w} = xyx \in \hat{W}$, then $|x| \leq |\hat{w}|/n$ and most conjugates of $\hat{w}$ contain the square xx. Then some word from W generates a word containing a square equivalent to xx (Lemma 6), a contradiction.

So, it remains to establish (U3'). Since k may be quite big, a direct computer search does not look feasible. We shrink the size of this search to a polynomial in n using the following technical lemma and then apply computer.

Lemma 9. *A subset $\hat{W}'$ of $\hat{W}$ satisfies the condition (U3') if*
(a) different elements of $\hat{W}'$ are not conjugates, and
(b) for any $\hat{w} \neq \hat{z} \in \hat{W}'$, the word $\mu(1_n, w^4 z^4)$ is ε-threshold.

The condition (b) can be checked in time polynomial in n and independent of k. In fact, this lemma states that if a sufficiently long factor occurs twice in the word $\hat{w}\hat{z}$, then these two words are conjugates. The check of conjugacy can be further simplified with the help of Lemma 7. Finally, this check can also be made in $O(n^p)$ time for some p.

We verified the conditions of Lemma 9 for $5 \leq n \leq 101$. For $n \geq 11$, the whole set W is (ℓ, ε)-uniform. For $n = 9$, we found an (ℓ, ε)-uniform subset of size $3n+1$. So, in this cases Theorem 1 is obtained through Corollary 2 (2). For $n = 7$, (ℓ, ε)-uniform subsets are much smaller, but still enough to prove Theorem 1 through Corollary 1 (2). For $n = 5$, the maximal (ℓ, ε)-uniform subsets are of size 6 for $\varepsilon = 0$ and even less for $\varepsilon > 0$; this is not enough to "descent" from $n + 2 = 7$ through Corollary 1 (2). Finally, we note that the upper bound 101 is the limit of interest rather than the limit of feasibility: the next step in this direction should be an analytic proof for this (or some other) construction and arbitrarily big alphabets.

References

1. Badkobeh, G., Crochemore, M.: Finite-repetition threshold for infinite ternary words. In: Proc. 8th Internat. Conf. Words 2011 (WORDS 2011). EPTCS, vol. 63, pp. 37–43 (2011)
2. Brandenburg, F.J.: Uniformly growing k-th power-free homomorphisms. Theoret. Comput. Sci. 23, 69–82 (1983)
3. Brinkhuis, J.: Non-repetitive sequences on three symbols. Quart. J. Math. Oxford 34, 145–149 (1983)
4. Carpi, A.: On Dejean's conjecture over large alphabets. Theoret. Comput. Sci. 385, 137–151 (2007)
5. Currie, J.D., Rampersad, N.: A proof of Dejean's conjecture. Math. Comp. 80, 1063–1070 (2011)
6. Dejean, F.: Sur un théorème de Thue. J. Combin. Theory. Ser. A 13, 90–99 (1972)
7. Kolpakov, R., Rao, M.: On the number of Dejean words over alphabets of 5, 6, 7, 8, 9 and 10 letters. Theoret. Comput. Sci. 412, 6507–6516 (2011)
8. Lothaire, M.: Combinatorics on Words, Encyclopedia of Mathematics and Its Applications, vol. 17. Addison-Wesley (1983)
9. Ochem, P.: A generator of morphisms for infinite words. RAIRO Inform. Théor. App. 40, 427–441 (2006)
10. Ochem, P.: Letter frequency in infinite repetition-free words. Theoret. Comput. Sci. 380, 388–392 (2007)
11. Pansiot, J.J.: A propos d'une conjecture de F. Dejean sur les répétitions dans les mots. Discrete Appl. Math. 7, 297–311 (1984)
12. Rao, M.: Last cases of Dejean's conjecture. Theoret. Comput. Sci. 412, 3010–3018 (2011)

13. Restivo, A., Salemi, S.: Overlap Free Words on Two Symbols. In: Perrin, D., Nivat, M. (eds.) Automata on Infinite Words. LNCS, vol. 192, pp. 198–206. Springer, Heidelberg (1985)
14. Shallit, J.: Simultaneous avoidance of large squares and fractional powers in infinite binary words. Internat. J. Found. Comp. Sci. 15, 317–327 (2004)
15. Shur, A.M., Gorbunova, I.A.: On the growth rates of complexity of threshold languages. RAIRO Inform. Théor. App. 44, 175–192 (2010)

Probabilistic Automata and Probabilistic Logic

Thomas Weidner

Institut für Informatik, Universität Leipzig, D-04009 Leipzig, Germany
weidner@informatik.uni-leipzig.de

Abstract. We present a monadic second-order logic which is extended by an expected value operator and show that this logic is expressively equivalent to probabilistic automata for both finite and infinite words. We give possible syntax extensions and an embedding of our probabilistic logic into weighted MSO logic. We further derive decidability results which are based on corresponding results for probabilistic automata.

1 Introduction

Probabilistic automata, introduced already by Rabin [19], form a flourishing field. Their applications range from speech recognition [20] over prediction of climate parameters [17] to randomized distributed systems [11]. For surveys of theoretical results see the books [18,5]. Recently, the concept of probabilistic automata has been transfered to infinite words by Baier and Grösser [1]. This concept led to further research [2,7,8,9,10,22].

Though probabilistic automata admit a natural quantitative behavior, namely the acceptance probability of each word, the main research interest has been towards qualitative properties (for instance the language of all words with positive acceptance probability). We consider the behavior of a probabilistic automaton as function mapping finite or infinite words to a probability value. In spite of the paramount success of Büchi's [4] and Elgot's [14] characterizations of recognizable languages by MSO logic, no logic characterization of the behavior of probabilistic automata has been found yet.

We solve this problem by defining a probabilistic extension of MSO logic. Our *probabilistic MSO (PMSO)* logic is obtained from classical MSO logic by adding a second-order expected value operator $\mathbb{E}_p X$. In the scope of this operator, formulas $x \in X$ are considered to be true with probability p. The semantics of the expected value operator is then defined as the expected value over all sets. We illustrate our logic by an example of a communication device with probabilistic behavior, which can be modeled in PMSO.

In our main result, we establish the desired coincidence of behaviors of probabilistic automata and semantics of probabilistic MSO sentences. Our proof also yields a characterization of probabilistically recognizable word functions in terms of classical recognizable languages and Bernoulli measures. We show that every PMSO formula admits a prenex normal form, which is similar to existential MSO. Furthermore we give possible syntax extensions which do not alter the expressiveness of PMSO. Weighted MSO is another quantitative extension of MSO logic.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 813–824, 2012.

As shown in [12], a restricted form of weighted MSO is expressively equivalent to weighted automata for finite and infinite words. For finite words, probabilistic automata can be viewed as special case of weighted automata. We give a direct embedding of PMSO into the restricted fragment of weighted MSO, thus obtaining PMSO as a special case of weighted MSO.

There are many decidability results already known for probabilistic automata on finite [15] and infinite words [2,9,10]. By the expressive equivalence of probabilistic automata and PMSO, we obtain these results also for PMSO. For instance, it is decidable for a given formula whether there is a finite word with positive or almost sure semantics. On the downside, it is undecidable for a given formula whether there is a finite word with semantics greater than some non-trivial cut point, or if there is an infinite word with semantics equal to one.

2 Bernoulli Measures and Probabilistic Automata

For the rest of this work let Σ be a finite alphabet. We let Σ^+ be the set of all finite, non-empty words over Σ, and Σ^ω the set of all infinite words over Σ. If $w \in \Sigma^+$ is a finite word we write $|w|$ for its length. If $w \in \Sigma^\omega$ we let $|w| = \omega$. For convenience we write Σ^∞ if either Σ^+ or Σ^ω can be used. For a word $w \in \Sigma^\infty$ let $\mathrm{dom}(w)$ be $\mathbb{N}$ if $w \in \Sigma^\omega$ and $\{1, \dots, |w|\}$ otherwise.

For two words $u = (u_i)_{i \in \mathrm{dom}(u)} \in \Sigma_1^\infty$ and $v = (v_i)_{i \in \mathrm{dom}(v)} \in \Sigma_2^\infty$ with $\mathrm{dom}(u) = \mathrm{dom}(v)$ we define the word (u, v) as $((u_i, v_i))_{i \in \mathrm{dom}(u)} \in (\Sigma_1 \times \Sigma_2)^\infty$.

Given a set X and a subset $Y \subseteq X$ let $\mathbb{1}_Y \colon X \to \{0, 1\}$ be the characteristic function of Y, i.e. $\mathbb{1}_Y(x) = 1$ if $x \in Y$ and $\mathbb{1}_Y(x) = 0$ otherwise. In the case $X = \mathbb{N}$, $\mathbb{1}_Y$ is also interpreted as ω-word over $\{0, 1\}$. Also for $f \colon X \to \mathbb{R}$ let $\mathrm{supp}(f) = \{x \in X \mid f(x) \neq 0\}$.

A *σ-field* over a set Ω is a system $\mathcal{A}$ of subsets of Ω which includes the empty set and is closed under complement and countable union. The pair $(\Omega, \mathcal{A})$ is called a *measurable space*. A *measure* on $\mathcal{A}$ is a mapping $\mu \colon \mathcal{A} \to [0, \infty]$ such that $\mu(\emptyset) = 0$ and $\mu(\bigcup_{i \geq 1} M_i) = \sum_{i \geq 1} \mu(M_i)$ for pairwise disjoint $M_i \in \mathcal{A}$. If $\mu(\Omega) = 1$, μ is called a *probability measure*.

Let $(\Omega', \mathcal{A}')$ be another measurable space. A function $f \colon \Omega \to \Omega'$ is $\mathcal{A}$-$\mathcal{A}'$-*measurable* if $f^{-1}(M') \in \mathcal{A}$ for every $M' \in \mathcal{A}'$. Now let f be $\mathcal{A}$-$\mathcal{A}'$-measurable and μ a measure on $\mathcal{A}$. The *image measure* of μ under f is the measure $\mu \circ f^{-1}$ on $\mathcal{A}'$ defined by $(\mu \circ f^{-1})(M') = \mu(f^{-1}(M'))$ for all $M' \in \mathcal{A}'$.

A measurable function $s \colon \Omega \to \mathbb{R}$ of the form $s = \sum_{i=1}^n r_i \mathbb{1}_{M_i}$ for $r_i \geq 0$ and $M_i \in \mathcal{A}$ is called simple. The integral of s is defined by $\int s \, \mathrm{d}\mu = \sum_{i=1}^n r_i \cdot \mu M_i$. For an arbitrary $\mathcal{A}$-Borel($\mathbb{R}$)-measurable function $f \colon \Omega \to [0, \infty]$ the integral is then given by

$$\int f \, \mathrm{d}\mu = \int_\Omega f(x) \, \mu(\mathrm{d}x) = \sup \left\{ \int s \, \mathrm{d}\mu \;\middle|\; 0 \leq s \leq f, \, s \text{ simple} \right\}.$$

2.1 Bernoulli Measures

Let X be a finite set. We denote the product σ-field $\bigotimes_{i=1}^\infty \mathcal{P}(X)$ on the base set X^ω by $\mathcal{A}_X$. Then $\mathcal{A}_X$ is the σ-field generated by all cones xX^ω for $x \in X^*$. As the system of all cones is closed under intersection, any two measures that agree on all cones are equal by standard measure theory.

For a probability distribution $p = (p_x)_{x\in X}$ on X let $\mathbb{P}_p^X$ be the corresponding probability measure on $\mathcal{P}(X)$, i.e. $\mathbb{P}_p^X(\{x\}) = p_x$ for all $x \in X$. If $X = \{0,1\}$ and $p \in [0,1]$, we simply write $\mathbb{P}_p$ for $\mathbb{P}^{\{0,1\}}_{(1-p,p)}$, i.e. $\mathbb{P}_p(\{1\}) = p$.

The *Bernoulli measure* $\mathrm{B}_p^{X^\omega}$ is defined as the product measure $\bigotimes_{i=1}^{\infty} \mathbb{P}_p^X$. It is explicitly given by $\mathrm{B}_p^{X^\omega}(x_1 \cdots x_k X^\omega) = \prod_{i=1}^{k} p_{x_i}$, for all cones $x_1 \cdots x_k X^\omega \in \mathcal{A}_X$. Hence $\mathrm{B}_p^{X^\omega}$ is a probability measure on $\mathcal{A}_X$. In the case $X = \{0,1\}$ and for $p \in [0,1]$ we write B_p^ω for $\mathrm{B}^{\{0,1\}^\omega}_{(1-p,p)}$. For more background information on Bernoulli- and product measures see for example [16, Theorem 1.64].

A binary sequence $m = (m_i)_{i\in \mathrm{dom}(m)} \in \{0,1\}^\infty$ corresponds bijectively to the set $\mathrm{supp}(m) = \{i \in \mathrm{dom}(m) \mid m_i = 1\}$. We define A_ω as the σ-field on $\mathcal{P}(\mathbb{N})$ generated by supp, i.e. $\mathcal{A}_\omega = \mathrm{supp}(\mathcal{A}_{\{0,1\}})$. Note that A_ω is actually a σ-field, as supp is bijective. The Bernoulli measure B_p^ω is also transfered to $\mathcal{A}_\omega$ by supp, i.e. $\mathrm{B}_p^\omega \circ \mathrm{supp}^{-1}$ is a measure on $\mathcal{A}_\omega$. We will denote this measure also by B_p^ω.

Let $n \in \mathbb{N}$. It is also possible to define a Bernoulli measure on the finite σ-field $\mathcal{P}(X^n)$. The measure $\mathrm{B}_p^{X^n}$ is defined as the finite product $\bigotimes_{i=1}^{n} \mathbb{P}_p^X$. The measure B_p^n on $\{0,1\}^n$ resp. $\mathcal{P}(\{1,\ldots,n\})$ is defined analogously to the infinite case: $\mathrm{B}_p^n = \mathrm{B}^{\{0,1\}^n}_{(1-p,p)}$ resp. $\mathrm{B}_p^n = \mathrm{B}^{\{0,1\}^n}_{(1-p,p)} \circ \mathrm{supp}^{-1}$.

2.2 Probabilistic Automata

A *probabilistic automaton* A is given by a quadruple (Q, δ, μ, F), where

- Q is a finite set of *states*
- $\delta\colon Q \times \Sigma \times Q \to [0,1]$ is the *transition probability function* such that $\sum_{q\in Q} \delta(r,a,q) = 1$ for every $r \in Q$ and $a \in \Sigma$
- $\mu\colon Q \to [0,1]$ is the *initial distribution* such that $\sum_{q\in Q} \mu(q) = 1$
- $F \subseteq Q$ is the set of *final states.*

For a word $w = w_1 \ldots w_k \in \Sigma^+$ we define the *behavior* $\|A\|\colon \Sigma^+ \to [0,1]$ *of* A by

$$\|A\|(w) := \sum_{\substack{q_0,\ldots,q_{k-1}\in Q\\ q_k\in F}} \mu(q_0) \prod_{i=1}^{k} \delta(q_{i-1}, w_i, q_i),$$

for each $w \in \Sigma^+$. It follows that $\|A\|(w) \in [0,1]$ for every $w \in \Sigma^+$.

We call a function $S\colon \Sigma^+ \to [0,1]$ *probabilistically recognizable* if there is a probabilistic automaton A such that $S = \|A\|$.

2.3 Probabilistic ω-Automata

Probabilistic ω-automata are a generalization of deterministic ω-automata. A *probabilistic Muller-automaton* A over an alphabet Σ is a quadruple $(Q, \delta, \mu, \mathcal{F})$, where Q, δ, μ are defined as in the finite case and $\mathcal{F} \subseteq \mathcal{P}(Q)$ is a Muller acceptance condition (cf. [1]).

For an infinite run $\rho \in Q^\omega$ let $\inf(\rho)$ denote the set of states which occur infinitely often in ρ. We say the run ρ is successful, if $\inf(\rho) \in \mathcal{F}$. For each word $w = w_1 w_2 \ldots \in \Sigma^\omega$ we define a probability measure $\mathbb{P}_A^w$ on the σ-field $\mathcal{A}_Q$ by

$$\mathbb{P}_A^w(q_0 \ldots q_k Q^\omega) := \mu(q_0) \cdot \prod_{i=1}^{k} \delta(q_{i-1}, w_i, q_i).$$

By standard measure theory, there is exactly one such probability measure $\mathbb{P}_A^w$.

The *behavior* $\|A\| \colon \Sigma^\omega \to [0,1]$ *of* A is then given by

$$\|A\|(w) := \mathbb{P}_A^w\big(\rho \in Q^\omega \ ; \ \inf(\rho) \in \mathcal{F}\big),$$

i.e. $\|A\|(w)$ is the measure of the set of all successful runs.

We call a function $S \colon \Sigma^\omega \to [0,1]$ *probabilistically ω-recognizable* if there is a probabilistic Muller-automaton A such that $S = \|A\|$.

3 Syntax and Semantics of PMSO

This section first introduces assignments and encodings. Using these definitions we will define the syntax and semantics of probabilistic MSO (PMSO) logic. Afterwards, we will give first semantic equivalences and consider possible syntax extensions.

3.1 Assignments and Encodings

For a uniform treatment of semantics we introduce assignments and their encodings. Let $\mathcal{V}_1$ be a finite set of first order variable symbols and $\mathcal{V}_2$ a disjoint finite set of second order variable symbols. We write $\mathcal{V}$ for $\mathcal{V}_1 \,\dot\cup\, \mathcal{V}_2$. Let $w \in \Sigma^\infty$ be a word. A mapping $\alpha \colon \mathcal{V} \to \mathrm{dom}(w) \cup \mathcal{P}(\mathrm{dom}(w))$ is called a $(\mathcal{V}, w)$-assignment if $\alpha(\mathcal{V}_1) \subseteq \mathrm{dom}(w)$ and $\alpha(\mathcal{V}_2) \subseteq \mathcal{P}(\mathrm{dom}(w))$. For $i \in \mathrm{dom}(w)$ and $x \in \mathcal{V}_1$ the assignment $\alpha[x \to i]$ denotes the $(\mathcal{V} \cup \{x\}, w)$-assignment which assigns x to i and agrees with α on all other variables. Likewise for $M \subseteq \mathrm{dom}(w)$ and $X \in \mathcal{V}_2$ the $(\mathcal{V} \cup \{X\}, w)$-assignment $\alpha[X \to M]$ assigns X to M and agrees with α everywhere else. We write $\alpha[L_1 \mapsto R_1, \ldots, L_n \mapsto R_n]$ for the chained assignment $\alpha[L_1 \mapsto R_1] \cdots [L_n \mapsto R_n]$.

We encode assignments as words as usual. The extended alphabet $\Sigma_\mathcal{V}$ is defined as $\Sigma \times \{0,1\}^\mathcal{V}$. Let $\overline{w} = ((w_i, \alpha_i))_{i \in \mathrm{dom}(\overline{w})} \in \Sigma_\mathcal{V}^\omega$ and $w = (w_i)_{i \in \mathrm{dom}(\overline{w})}$. We say $\overline{w}$ encodes an $(\mathcal{V}, w)$-assignment α if for every $x \in \mathcal{V}_1$ there is exactly one position j such that $\alpha_j(x) = 1$. In this case $\alpha(x)$ is then the unique position i with $\alpha_i(x) = 1$ and, for $X \in \mathcal{V}_2$, $\alpha(X)$ is the set of all positions j' with $\alpha_{j'}(X) = 1$. We denote the set of all valid encodings by $\mathcal{N}_\mathcal{V} \subseteq \Sigma_\mathcal{V}^\infty$.

Likewise every pair of a word $w \in \Sigma^\infty$ and a $(\mathcal{V}, w)$-assignment α can be encoded as a word in $N_\mathcal{V}$ in the obvious way. We will use (w, α) to describe both the pair and its encoding as word depending on the context.

Table 1. PMSO semantics

$$[\![\mathrm{P}_a(x)]\!](w,\alpha) = \begin{cases} 1, & \text{if } w_{\alpha(x)} = a \\ 0, & \text{otherwise} \end{cases} \qquad [\![x \le y]\!](w,\alpha) = \begin{cases} 1, & \text{if } \alpha(x) \le \alpha(y) \\ 0, & \text{otherwise} \end{cases}$$

$$[\![x \in X]\!](w,\alpha) = \begin{cases} 1, & \text{if } \alpha(x) \in \alpha(X) \\ 0, & \text{otherwise} \end{cases} \qquad \begin{aligned} [\![\neg\varphi]\!](w,\alpha) &= 1 - [\![\varphi]\!](w,\alpha) \\ [\![\varphi_1 \wedge \varphi_2]\!](w,\alpha) &= [\![\varphi_1]\!](w,\alpha) \cdot [\![\varphi_2]\!](w,\alpha) \end{aligned}$$

$$[\![\forall x.\varphi]\!](w,\alpha) = \begin{cases} 1, & \text{if } [\![\varphi]\!](w,\alpha[x \to i]) = 1 \text{ for all } i \in \mathrm{dom}(w) \\ 0, & \text{otherwise} \end{cases}$$

$$[\![\forall X.\varphi]\!](w,\alpha) = \begin{cases} 1, & \text{if } [\![\varphi]\!](w,\alpha[X \to M]) = 1 \text{ for all } M \subseteq \mathrm{dom}(w) \\ 0, & \text{otherwise} \end{cases}$$

$$[\![\mathbb{E}_p X.\varphi]\!](w,\alpha) = \int_{\mathcal{P}(\mathrm{dom}(w))} [\![\varphi]\!](w,\alpha[X \mapsto M]) \; \mathrm{B}_p^{|w|}(\mathrm{d}M)$$

3.2 Boolean PMSO

Following an idea from [3], we define the syntax of *Boolean PMSO* (bPMSO) by

$$\psi ::= \mathrm{P}_a(x) \mid x \in X \mid x \le y \mid \psi \wedge \psi \mid \neg\psi \mid \forall x.\psi \mid \forall X.\psi,$$

for $x, y \in \mathcal{V}_1$, $X \in V_2$ and $a \in \Sigma$.

The set $\mathrm{Free}(\psi)$ of *free variables* in ψ is defined as usual.

The semantics $[\![\psi]\!]$ of a Boolean PMSO formula ψ maps a pair (w, α) of a word $w \in \Sigma^\infty$ and a $(\mathcal{V}, w)$-assignment α with $\mathrm{Free}(\psi) \subseteq \mathcal{V}$ to a value in $[0,1]$. The inductive definition of the semantics is given in the upper part of Table 1. It easily follows by structural induction that $[\![\psi]\!](w,\alpha) \in [0,1]$.

Boolean PMSO corresponds essentially to the classical MSO. Disjunction and existential quantification can be obtained from the defined operators as usual.

3.3 Full PMSO

We will now extend Boolean PMSO to full PMSO. The syntax of a PMSO formula φ is given in BNF by

$$\varphi ::= \psi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathbb{E}_p X.\varphi,$$

where ψ is a Boolean PMSO formula, $X \in \mathcal{V}_2$, and $p \in [0,1]$ is a real number. In other words, we have added an "expected value" operator $\mathbb{E}_p$ to Boolean PMSO and permit conjunction, negation and expected value as logical operations. The set of free variables of the expected value operator is

$$\mathrm{Free}(\mathbb{E}_p X.\varphi) := \mathrm{Free}(\varphi) \setminus \{X\}\,.$$

The semantics of a PMSO formula is given in the full Table 1.

In order for $[\![\mathbb{E}_p X.\varphi]\!]$ to be well-defined, one must and can show, that the function $M \mapsto [\![\varphi]\!](w, \alpha[X \mapsto M])$ is $\mathcal{A}_\omega$-measurable and integrable for all (w, α). This is a consequence of the measurability of all ω-recognizable sets (cf. [3,23]) and of Fubini's theorem.

In case of finite words, the semantics of $\mathbb{E}_p X.\varphi$ can be rewritten to

$$[\![\mathbb{E}_p X.\varphi]\!](w, \alpha) = \sum_{M \subseteq \mathrm{dom}(w)} [\![\varphi]\!](w, \alpha[X \mapsto M]) \cdot p^{|M|}(1-p)^{|w|-|M|}.$$

Next we give an intuitive argument for the semantics of $\mathbb{E}_p X.\varphi$. The classical existential quantifier states the existence of a set which satisfies the quantified formula. Here, for the expected value operator sets are chosen using a stochastic process: For every position k we make a probabilistic choice whether k should be included in the set or not, where the probability of inclusion is p. This choice is independent from the other positions. Such a process can be considered as tossing an unfair coin for every position k to decide whether $k \in X$ holds. The semantics of the expected value operator is then the expected value of $[\![\varphi]\!]$ under this distribution. If φ is Boolean, this value can be considered as the probability that $(w, \alpha[X \mapsto M])$ satisfies φ for an arbitrary set M.

The semantics of a PMSO formula φ transfers to the extended alphabet $\Sigma_\mathcal{V}$ with $\mathrm{Free}(\varphi) \subseteq \mathcal{V}$ as follows. We define $[\![\varphi]\!]_\mathcal{V} \colon \Sigma_\mathcal{V}^\infty \to \mathbb{R}$ by

$$[\![\varphi]\!]_\mathcal{V}(\overline{w}) := \begin{cases} [\![\varphi]\!](w, \alpha), & \text{if } \overline{w} \in \mathcal{N}_\mathcal{V} \text{ and } \overline{w} = (w, \alpha) \\ 0, & \text{otherwise.} \end{cases}$$

We will use some common abbreviations:

$$(\varphi \vee \psi) := \neg(\neg\varphi \wedge \neg\psi), \qquad (\varphi \to \psi) := \neg(\varphi \wedge \neg\psi),$$
$$(\exists x.\eta) := \neg\forall x.\neg\eta, \qquad (\exists X.\eta) := \neg\forall X.\neg\eta,$$

for formulas $\varphi, \psi \in$ PMSO and $\eta \in$ bPMSO. Note that if φ and ψ are Boolean PMSO formulas, then the abbreviated formulas are again Boolean.

From the definition the semantics of $\varphi \vee \psi$ is $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] + [\![\psi]\!] - [\![\varphi]\!][\![\psi]\!]$. This is analogous to the fact that that the probability of the union of two independent events A and B is $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A)\mathbb{P}(B)$.

The following example demonstrates the use of PMSO logic using a model of a communication device.

Example 1. We consider a communication device for sending messages. At every point of time either a new input message becomes available or the device is waiting for a new message. When a new message is available the device tries to send this message. Sending a message may fail with probability 1/3. In this case the message is stored in an internal buffer. The next time the device is waiting for a message, sending the stored message is retried. Intuitively, as sending a buffered message has already failed once, it seems to be harder to send this message. So sending a buffered message is only successful with probability 1/2. The buffer can hold one message.

The PMSO sentence below defines for every sequence of message input (i) and wait (w) cycles the probability that this sequence will not overflow the device's buffer. In this sentence the set variable I contains all positions (i.e. points of time) where sending an input message is successful, B all positions where sending a buffered message is successful, and F all positions where the buffer is full.

$$\begin{aligned}&\mathbb{E}_{2/3} I.\mathbb{E}_{1/2} B.\exists F.1 \notin F \wedge \forall x.\forall y.y = x+1 \rightarrow \\ &\big((\mathrm{P}_\mathrm{w}(x) \wedge x \notin B) \rightarrow (x \in F \leftrightarrow y \in F)\big) \wedge \big((\mathrm{P}_\mathrm{w}(x) \wedge x \in B) \rightarrow y \notin F\big) \wedge \\ &\quad\big((P_\mathrm{i}(x) \wedge x \in I) \rightarrow (x \in F \leftrightarrow y \in F)\big) \wedge \big((\mathrm{P}_\mathrm{i}(x) \wedge x \notin I) \rightarrow (x \notin F \wedge y \in F)\big)\end{aligned}$$

3.4 Basic Properties of PMSO Semantics

The following consistency lemma is a fundamental property.

Lemma 1. *Let φ be a PMSO formula, $w \in \Sigma^\infty$, $\mathcal{V}$ a finite set of variables such that* $\mathrm{Free}(\varphi) \subseteq \mathcal{V}$*, and α a $(\mathcal{V}, w)$-assignment. Then* $[\![\varphi]\!](w, \alpha) = [\![\varphi]\!](w, \alpha|_{\mathrm{Free}(\varphi)})$.

As usual we call PMSO formula φ a PMSO sentence if $\mathrm{Free}(\varphi) = \emptyset$. As a consequence of Lemma 1, if φ is a PMSO sentence, we define $[\![\varphi]\!](w)$ as $[\![\varphi]\!](w, \alpha)$ where α is an arbitrary $(\mathcal{V}, w)$-assignment.

For two PMSO formulas φ and ψ we write $\varphi \equiv \psi$, if $[\![\varphi]\!] = [\![\psi]\!]$ holds. It follows from the semantics definition that the usual associativity, commutativity, and distributivity laws also hold for PMSO logic. For distributivity the outer formula has to be a Boolean one, i.e. $\varphi \wedge (\psi_1 \vee \psi_2) \equiv (\varphi \wedge \psi_1) \vee (\varphi \wedge \psi_2)$ only if $\varphi \in \mathrm{bPMSO}$. For formulas containing the expected value operator, we obtain new equivalences:

$$\begin{aligned}\neg\mathbb{E}_p X.\varphi &\equiv \mathbb{E}_p X.\neg\varphi, & \mathbb{E}_p X.(\varphi \wedge \psi) &\equiv \varphi \wedge \mathbb{E}_p X.\psi \text{ if } X \notin \mathrm{Free}(\varphi),\\ \mathbb{E}_p X.\mathbb{E}_q Y.\varphi &\equiv \mathbb{E}_q Y.\mathbb{E}_p X.\varphi, & \mathbb{E}_p X.\varphi &\equiv \varphi \text{ if } X \notin \mathrm{Free}(\varphi).\end{aligned}$$

Note that contrary to classical quantifiers, pulling negation out of the expected value operator does not change the operator at all.

These equivalences allow us to transform PMSO formulas to a simpler form. We say a formula φ is in *prenex normal form* if it of the form

$$\varphi = \mathbb{E}_{p_1} X_1 \dots \mathbb{E}_{p_k} X_k.\varphi_0$$

for a bPMSO formula φ_0, real values $p_1, \dots, p_k \in [0, 1]$, and distinct second order variables $X_1, \dots, X_k$.

Lemma 2. *Let φ be a PMSO formula, then there is an equivalent PMSO formula φ' in prenex normal form.*

3.5 Syntax Extensions

We discuss three possible syntax extensions in this section. extensions do not alter the expressiveness of PMSO.

Probability Constants. For a real number $p \in [0,1]$, we add the formula "p" to PMSO and define the semantics by $[\![p]\!](w,\alpha) = p$ for all $w \in \Sigma^\infty$ and all $(\mathcal{V}, w)$-assignments α. Then p can be expressed in PMSO by $\mathbb{E}_p X.1 \in X$.

An Extended First Order Universal Quantifier. As for weighted MSO (see Section 5), it is possible to extend the syntax and semantics of the universal first order quantifier in PMSO to PMSO formulas φ by

$$[\![\forall x.\varphi]\!](w,\alpha) := \prod_{i \in \mathrm{dom}(w)} [\![\varphi]\!](w, \alpha[x \mapsto i]).$$

Unfortunately it follows using a shrinkage argument that this form of the universal quantifier does not preserve recognizability. Therefore we restrict φ to particular formulas, which we define next.

A *step formula* is a PMSO formula φ such that there are bPMSO formulas $\varphi_1, \ldots, \varphi_n$ and real numbers $p_1, \ldots, p_n \in [0,1]$ such that φ is equivalent to $\bigwedge_{i=1}^n (\varphi_i \to p_i)$. When this condition is satisfied $\forall x.\varphi$ can be rewritten as

$$\mathbb{E}_{p_1} X_1 \ldots . \mathbb{E}_{p_n} X_n . \forall x. \bigwedge_{i=1}^n (\varphi_i \to x \in X_i),$$

for new second order variables $X_1, \ldots, X_n$.

A First Order Expected Value Operator. In the case of infinite words, it is possible to define a first order expected value operator with reasonable semantics.

Let φ be a PMSO formula, $p \in (0,1)$, and x a first order variable. We define

$$\mathbb{E}_p x.\varphi := \mathbb{E}_p X.\tilde{\varphi},$$

where $\tilde{\varphi}$ is obtained from φ by replacing every occurrence of x with $\min X$. Note that, though "$\min X$" is not valid PMSO syntax, it is a well-known MSO property and thus expressible in PMSO. Also $\{\emptyset\}$ is a B_p^ω-null set.

To express the semantics of the just defined operator in a natural way, we introduce the geometric distribution on $\mathbb{N}$. For $p \in (0,1)$ let $\mathrm{G}_p\big(\{n\}\big) = (1-p)^{n-1}p$. Intuitively, $\mathrm{G}_p(\{n\})$ is the probability to get one success after n experiments in an infinitely running Bernoulli experiment. It follows that $\mathrm{G}_p = \mathrm{B}_p^\omega \circ \min^{-1}$. We apply this equality to $\mathbb{E}_p x.\varphi$ and obtain

$$[\![\mathbb{E}_p x.\varphi]\!](w,\alpha) = \int_{\mathbb{N}} [\![\varphi]\!](w, \alpha[x \mapsto i]) \; \mathrm{G}_p(\mathrm{d}i).$$

4 Equivalence of PMSO and Probabilistic Automata

Our main theorem establishes the desired expressive equivalence of PMSO sentences and probabilistic automata for both cases of finite and infinite words.

Theorem 1. *Let Σ be an alphabet.*

1. *A function $S\colon \Sigma^+ \to [0,1]$ is probabilistically recognizable iff there is a PMSO sentence φ such that $[\![\varphi]\!] = S$.*
2. *A function $S\colon \Sigma^\omega \to [0,1]$ is probabilistically ω-recognizable iff there is a PMSO sentence φ such that $[\![\varphi]\!] = S$.*

We will sketch the proof of Theorem 1 for infinite words in the rest of this section. The finite case is analogous.

For the rest of the section we write simply B_p for the Bernoulli measure if the base set is understood.

4.1 Characterization by Bernoulli Measures

We give a characterization of probabilistically recognizable functions using Bernoulli measures. This characterization could be of independent interest.

Theorem 2. *A function $S\colon \Sigma^\omega \to [0,1]$ is probabilistically ω-recognizable iff there is an alphabet Γ, a distribution p on Γ and an ω-recognizable language $L \subseteq (\Sigma \times \Gamma)^\omega$ such that*

$$S(w) = \mathrm{B}_p\big(\{u \in \Gamma^\omega \mid (w,u) \in L\}\big), \tag{1}$$

for all $w \in \Sigma^\omega$.

Equation 1 means that S is an image measure. Indeed if $\theta_w(u) = (w,u)$, then (1) can be written as $S(w) = \mathrm{B}_p \circ \theta_w^{-1}(L)$.

Proof. Given a probabilistic ω-automaton $A = (Q, \delta, \mathbb{1}_{\{\iota\}}, \mathcal{F})$ we use an enumeration $0 = d_0 < \ldots < d_n = 1$ of the set $\{\sum_{i=1}^{q} \delta(p,a,q) \mid p,q \in Q,\ a \in \Sigma\} \cup \{0\}$ to define $\Gamma{:=}\{1,\ldots,n\}$ and p by $p_k{:=}d_k - d_{k-1}$ for all $k \in \Gamma$. We construct a Muller-automaton $B = (Q, T, \iota, \mathcal{F})$ from A such that $\sum_{u\in\Gamma,\,(p,(a,u),q)\in T} = \delta(p,a,q)$ and define L as the language accepted by B.

Conversely, given a Muller-automaton $B = (Q, T, \iota, \mathcal{F})$, we construct a probabilistic Muller-automaton A which recognizes S. We define $A := (Q, \delta, \mathbb{1}_{\{\iota\}}, \mathcal{F})$ where $\delta(p,a,q) := \sum_{u\in\Gamma,\,(p,(a,u),q)\in T} p_u$. □

The last theorem used a Bernoulli measure on a finite, but arbitrary large, set Γ. In PMSO only Bernoulli measures on the two element set $\{0,1\}$ are available.

Corollary 1. *A function $S\colon \Sigma^\omega \to [0,1]$ is probabilistically ω-recognizable iff there are a natural number $n \in \mathbb{N}$, real numbers $r_1,\ldots,r_n \in [0,1]$ and an ω-recognizable language $L \subseteq (\Sigma \times \{0,1\}^n)^\omega$ such that*

$$S(w) = \left(\bigotimes_{i=1}^{n} \mathrm{B}_{r_i}\right)\big(\{(M_1,\ldots,M_n) \in \mathcal{P}(\mathbb{N})^n \mid (w, \mathbb{1}_{M_1},\ldots,\mathbb{1}_{M_n}) \in L\}\big),$$

for all $w \in \Sigma^\omega$.

Proof. We show that a Bernoulli measure on an arbitrary finite set can be written as an image measure under a suitable mapping h of a finite product of binary Bernoulli measures. Next, we apply Theorem 2 and show that h^{-1} retains the recognizability of L in Theorem 2. □

4.2 Proof of Theorem 1

Let $S\colon \Sigma^\omega \to [0,1]$ be probabilistically ω-recognizable. By Corollary 1 there are $n \in \mathbb{N}$, real numbers $r_1, \ldots, r_n \in [0,1]$ and an ω-recognizable language $L \subseteq (\Sigma \times \{0,1\}^n)^\omega$ such that

$$S(w) = \left(\bigotimes_{i=1}^{n} \mathrm{B}_{r_i}\right)\left(\left\{(M_1, \ldots, M_n) \in \mathcal{P}(\mathbb{N})^n \;\middle|\; (w, \mathbb{1}_{M_1}, \ldots, \mathbb{1}_{M_n}) \in L\right\}\right).$$

Let $\mathcal{V} = \{X_1, \ldots, X_n\}$. By Büchi's theorem there is a bPMSO formula φ_0 over Σ with $\mathrm{Free}(\varphi_0) = \mathcal{V}$ which defines L, i.e. $L = \mathrm{supp}([\![\varphi_0]\!]_\mathcal{V})$. Let φ be the PMSO sentence given by

$$\varphi = \mathbb{E}_{r_1} X_1. \ldots \mathbb{E}_{r_n} X_n.\varphi_0.$$

It follows by Fubini's theorem that $S = [\![\varphi]\!]$.

Conversely, let φ be a PMSO sentence with $S = [\![\varphi]\!]$. By Lemma 2 we may assume that φ is in prenex normal form, i.e. $\varphi = \mathbb{E}_{r_1} X_1. \ldots \mathbb{E}_{r_n} X_n.\varphi_0$ for a Boolean PMSO formula φ_0, real numbers $r_1, \ldots, r_n \in [0,1]$, and distinct set variables $X_1, \ldots, X_n$. Let $\mathcal{V} = \{X_1, \ldots, X_n\}$ and $L = \mathrm{supp}([\![\varphi_0]\!]_\mathcal{V})$. Then $[\![\varphi_0]\!]_\mathcal{V} = \mathbb{1}_L$ and L is ω-recognizable by Büchi's theorem as φ_0 is Boolean. By Fubini's theorem we obtain

$$S(w) = \left(\bigotimes_{i=1}^{n} \mathrm{B}_{r_i}\right)\left(\left\{(M_1, \ldots, M_n) \in \mathcal{P}(\mathbb{N})^n \;\middle|\; (w, \mathbb{1}_{M_1}, \ldots, \mathbb{1}_{M_n}) \in L\right\}\right).$$

Therefore S is probabilistically ω-recognizable by Corollary 1. □

Remark 1. When translating a PMSO formula to a probabilistic Muller-automaton the acceptance condition of the automaton can be chosen to be a Rabin, Streett, or parity condition. This is because for all of these acceptance conditions classical ω-automata can be determinized. The latter is not true for the Büchi, reachability or safety acceptance conditions.

5 Relation to Weighted MSO

In [12] a weighted MSO (wMSO) logic was introduced. It was shown that a certain fragment of weighted MSO logic is expressively equivalent to weighted automata. This expressive equivalence holds for finite and infinite words and also for arbitrary semirings. Whereas probabilistic automata on infinite words represent a different model than weighted automata on infinite words, probabilistic automata on finite words are a special case of weighted automata over the semiring of the non-negative real numbers $\mathbb{R}^+$.

For the exact definitions of weighted automata, weighted MSO, and syntactically restricted weighted MSO (srMSO) see [12,13].

As shown in [12], a function $S\colon \Sigma^+ \to \mathbb{R}^+$ is recognizable by a weighted automaton iff S is definable in srMSO. Hence every PMSO formula can be translated to a probabilistic automaton, which then can be translated to a srMSO formula. We give a direct mapping to embed PMSO into srMSO using a syntactic transformation.

Theorem 3. *Let φ be a PMSO formula. Then there is a srMSO formula φ' over the semiring of the non-negative real numbers such that $[\![\varphi]\!] = [\![\varphi']\!]$. Moreover φ' can be obtained from φ by a effective syntactic transformation.*

Intuitively, φ' is obtained from φ by replacing every occurrence of $\mathbb{E}_p X.\varphi_0$ with $\exists X.\varphi_0 \wedge \forall x.\big((r \wedge x \in X) \vee ((1-r) \wedge x \notin X)\big)$ where x is a new variable.

6 Conclusion and Future Work

We introduced a probabilistic extension of classical MSO logic by the addition of an expected value operator. We could show that, similarly to the fundamental Büchi-Elgot-Theorem, this probabilistic MSO logic is expressively equivalent to probabilistic automata on finite and infinite words. We also gave several syntax extensions and an effective embedding into weighted MSO logic.

As our transformations between PMSO sentences and probabilistic automata are effective, all decidability results for probabilistic automata also apply to PMSO sentences. For example, it is decidable for a PMSO sentence φ if there is a finite word $w \in \Sigma^+$ such that $[\![\varphi]\!](w) > 0$ $(= 1)$ [18], or if two given PMSO formulas are equivalent on finite words [21]. On the other hand, interesting problems are undecidable. For instance, for a given formula φ it is undecidable if there is an infinite word w such that $[\![\varphi]\!](w) > 0$ $(= 1)$ [2]. Another undecidable problem is to decide for a formula φ and some $\lambda \in (0, 1)$ if there is a finite or infinite word w such that $[\![\varphi]\!](w) > \lambda$ [18]. This problem remains undecidable even for $\lambda = {}^1\!/\!_2$ and $\varphi = \mathbb{E}_{1/2} X.\varphi_0$ where φ_0 is Boolean [15].

Many concepts of probabilistic ω-automata like safety, reachability or Büchi acceptance conditions, hierarchical probabilistic automata [6], #-acyclic automata [15], or probabilistic automata which induce a simple process [10] have better decidability properties. It is an open problem to derive any of these concepts for PMSO logic.

In current work, we wish to find similar probabilistic extensions for temporal logics. For example a suitable probabilistic LTL should be expressively equivalent to the first order fragment of probabilistic MSO logic. We also hope to obtain better decidability properties using this approach.

Acknowledgements. This work is part of ongoing PhD work supervised by Prof. Manfred Droste at Leipzig University.

References

1. Baier, C., Grösser, M.: Recognizing ω-regular languages with probabilistic automata. In: Proc. LICS, pp. 137–146. IEEE (2005)
2. Baier, C., Bertrand, N., Größer, M.: On Decision Problems for Probabilistic Büchi Automata. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 287–301. Springer, Heidelberg (2008)
3. Bollig, B., Gastin, P.: Weighted versus Probabilistic Logics. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 18–38. Springer, Heidelberg (2009)

4. Büchi, J.R.: Weak second-order arithmetic and finite automata. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 6, 66–92 (1960)
5. Bukharaev, R.G.: Theorie der stochastischen Automaten. Teubner (1995)
6. Chadha, R., Sistla, A.P., Viswanathan, M.: Power of Randomization in Automata on Infinite Strings. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 229–243. Springer, Heidelberg (2009)
7. Chadha, R., Sistla, A.P., Viswanathan, M.: Probabilistic Büchi Automata with Non-extremal Acceptance Thresholds. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 103–117. Springer, Heidelberg (2011)
8. Chatterjee, K., Doyen, L., Henzinger, T.A.: Probabilistic Weighted Automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 244–258. Springer, Heidelberg (2009)
9. Chatterjee, K., Henzinger, T.: Probabilistic Automata on Infinite Words: Decidability and Undecidability Results. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 1–16. Springer, Heidelberg (2010)
10. Chatterjee, K., Tracol, M.: Decidable problems for probabilistic automata on infinite words. CoRR abs/1107.2091 (2011)
11. Cheung, L., Lynch, N., Segala, R., Vaandrager, F.: Switched PIOA: Parallel composition via distributed scheduling. TCS 365(1-2), 83–108 (2006)
12. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theoretical Computer Science 380(1-2), 69–86 (2007)
13. Droste, M., Kuich, W., Vogler, H., (eds.): Handbook of Weighted Automata. EATCS Monographs in Theoretical Computer Science. Springer (2009)
14. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Amer. Math. Soc. 98, 21–51 (1961)
15. Gimbert, H., Oualhadj, Y.: Probabilistic Automata on Finite Words: Decidable and Undecidable Problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part II. LNCS, vol. 6199, pp. 527–538. Springer, Heidelberg (2010)
16. Klenke, A.: Probability Theory: A Comprehensive Course, 1st edn. Universitext. Springer (December 2007)
17. Mora-López, L., Morales, R., Sidrach de Cardona, M., Triguero, F.: Probabilistic finite automata and randomness in nature: a new approach in the modelling and prediction of climatic parameters. In: Proc. International Environmental Modelling and Software Congress, pp. 78–83 (2002)
18. Paz, A.: Introduction to Probabilistic Automata. Computer Science and Applied Mathematics. Academic Press, Inc. (1971)
19. Rabin, M.O.: Probabilistic automata. Information and Control 6(3), 230–245 (1963)
20. Ron, D., Singer, Y., Tishby, N.: The power of amnesia: Learning probabilistic automata with variable memory length. Machine Learning 25, 117–149 (1996)
21. Sakarovitch, J.: Rational and recognisable power series. In: Droste, et al. (eds.) [13] Handbook, pp. 105–174
22. Tracol, M., Baier, C., Größer, M.: Recurrence and transience for probabilistic automata. In: FSTTCS, vol. 4, pp. 395–406. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2009)
23. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite state programs. In: Foundations of Computer Science, pp. 327–338 (1985)

A Quadratic Vertex Kernel for Feedback Arc Set in Bipartite Tournaments

Mingyu Xiao[1,*] and Jiong Guo[2,**]

[1] School of Computer Science and Engineering,
University of Electronic Science and Technology of China,
Chengdu China
myxiao@gmail.com

[2] Universität des Saarlandes,
Campus E 1.7, D-66123 Saarbrücken, Germany
jguo@mmci.uni-saarland.de

Abstract. The k-feedback arc set problem is to determine whether there is a set F of at most k arcs in a directed graph G such that the removal of F makes G acyclic. The k-feedback arc set problems in tournaments and bipartite tournaments (k-FAST and k-FASBT) have applications in ranking aggregation and have been extensively studied from the viewpoint of parameterized complexity. Recently, Misra *et al.* provide a problem kernel with $O(k^3)$ vertices for k-FASBT. Answering an open question by Misra *et al.*, we improve the kernel bound to $O(k^2)$ vertices by introducing a new concept called "bimodule."

Keywords: Kernelization; Feedback arc set, Bipartite tournament, Graph algorithms, Parameterized algorithms.

1 Introduction

The *feedback set* problem, to make a graph acyclic by deleting minimum number (weight) of vertices or edges, is a classical graph problem and has applications in many areas. Several different versions of the problem, according to vertex or edge deletion, in directed or undirected graphs, and so on, have been extensively studied in the literature. The *feedback edge set* problem (edge deletion version) in undirected graphs is equivalent to the *maximum spanning tree* problem. However the *feedback vertex set* problem (vertex deletion version) in undirected graph is a famous NP-hard problem. When the input graph is a directed graph, both versions (deleting vertices or arcs) are polynomial-time reducible to each other and remain NP-hard. There is a long list of papers that study different versions of the feedback set problem concerning parameterized complexity. The corresponding parameterized problem—the *k-feedback set* problem—asks whether one can make a given graph acyclic by deleting as most k vertices or edges. We study the

[*] Supported by Grant 60903007 of NSFC, China.

[**] Supported by the DFG Excellence Cluster MMCI.

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 825–835, 2012.

k-feedback arc set problem in bipartite tournaments (orientations of complete bipartite graphs).

The k-feedback arc/vertex set problems in tournaments (orientations of complete graphs) and bipartite tournaments have applications in rank aggregation. A natural consistency measure for rank aggregation is the number of pairs that occur in a different order in the outcome ranking. This leads to *Kemeny rank aggregation* [11,12], a special case of a weighted version of the k-feedback arc set problem in tournaments. The problems in bipartite tournaments have similar applications, such as the establishment of mappings between ontologies (see [15] for more details).

In the last few years several interesting algorithmic results on the feedback set problem in tournaments and bipartite tournaments were achieved. Speckenmeyer [16] established the NP-completeness of the *k-feedback vertex set* problem in tournaments (k-FVST) many years ago. However the NP-completeness for k-FAST had been conjectured for a long time. Later, Alon [2] and Charbit *et al.* [6] independently proved this result. Cai *et al.* [4] showed that the k-feedback vertex set problem in bipartite tournaments is NP-complete. Finally, Guo *et al.* [8] proved that k-FASBT is also NP-complete. In terms of parameterized algorithms, k-FVST can be reduced to the 3-hitting set problem and then can be solved in $O^*(2.076^k)$ time [18]. Raman and Saurabh [19] showed that k-FAST is *fixed-parameter tractable* by giving an $O^*(2.415^k)$-time algorithm. Alon *et al.* [3] proved that k-FAST allows sub-exponential parameterized algorithms of running time $O^*(k^{O(\sqrt{k})})$, and Karpinsky and Schudy [10] improved the running time bound to $O^*(2^{O(\sqrt{k})})$. Sasatte [17] showed that k-FVSBT can be solved in $O^*(3^k)$ time, which was improved to $O^*(2^k)$ recently by Hsiao [9]. For k-FASBT, Dom *et al.* [7] gave an $O(3.373^k)$-time algorithm.

Kernelization is one of the most active subfields in parameterized complexity, in which we are asked to find a polynomial-time algorithm to reduce the input instance to an equivalent instance of size bounded by a function $p(k)$ of the parameter k. The resulted instance is also called a $p(k)$ *kernel* (or a *kernel of size* $p(k)$) of the problem. Different versions of the k-feedback set problem have been studied concerning kernelizations. It is known that k-FVST allows an $O(k^2)$-vertex kernel [1], k-FAST allows an $O(k)$-vertex kernel [5,14], and k-FVSBT allows an $O(k^3)$-vertex kernel [1]. Recently, Misra *et al.* [13] gave the first polynomial kernel of $O(k^3)$ vertices for k-FASBT and asked for kernels of $O(k^2)$ vertices. In this paper, we answer this open question affirmatively by presenting a kernel with $12k^2 - 2$ vertices. Our kernel algorithm is based on a newly introduced concept "bimodule." Previously, modules were used in [5,13] to get kernels for k-FAST and k-FASBT.

2 Preliminaries

A directed graph is denoted as $D = (V, A)$ and a bipartite tournament is denoted as $H = (X \cup Y, A)$. A subset F of arcs in a directed graph D is called a *feedback arc set* of D if the removal of F makes D acyclic. A feedback arc set is *minimal* if none of its proper subsets is a feedback arc set in the graph. Among all feedback arc sets, the ones of the minimum size are called *minimum feedback arc sets.* In k-FASBT, we are asked to check whether there is a feedback arc set of size at most k in a bipartite tournament. An instance of k-FASBT is denoted as (H, k). For a subset F' of arcs in a directed graph D, the graph $D\{F'\}$ is the directed graph obtained from D by reversing all arcs in F', while $D \setminus F'$ is the resulting graph after deleting F' from D. For a vertex v in $D = (V, A)$, we define $N^+(v) = \{w \in V | (v, w) \in A\}$ and $N^-(v) = \{u \in V | (u, v) \in A\}$. A vertex is *adjacent* to another vertex v if it is in $N^+(v) \cup N^-(v)$. For a directed graph D, we use $V(D)$ and $A(D)$ to denote the set of vertices and arcs in D respectively. For two disjoint vertex subsets V_1 and V_2, we use $A(V_1, V_2)$ to denote the set of all arcs with one endpoint in V_1 and the other one in V_2. For two subsets $S_1 \subseteq X$ and $S_2 \subseteq Y$ of a bipartite tournament $H = (X \cup Y, A)$, the bipartite tournament induced by $S_1 \cup S_2$ is denoted as $H[S_1 \cup S_2]$. When we say a topological sort of an acyclic graph, it means a linear ordering of the vertices of the graph where all arcs go from left to right. A cycle (resp. path) in a directed graph means a directed cycle (resp. directed path). A cycle (resp. path) with i vertices is called a *length-i cycle* (resp. *length-i path*). Sometimes a set $\{a\}$ of a single element is simply written as a. The following proposition is a folklore.

Proposition 1. *If a subset F of arcs in a directed graph $D = (V, A)$ is a minimal feedback arc set of D, then $D\{F\}$ is a directed acyclic graph.*

3 Bimodules and Their Properties

In order to present our kernelization algorithm, we firstly introduce a new concept called *bimodule* and analyze its properties. We will concentrate on some special vertices in a bimodule, called *k-central* and *k-redundant* vertices.

3.1 Bimodules of Bipartite Tournaments

We call two vertices u, v in H *similar*, if $N^-(u) = N^-(v)$ and $N^+(u) = N^+(v)$. A maximal set of pairwise similar vertices is called a *module.* A *bimodule* B of a bipartite tournament $H = (X \cup Y, A)$ is a pair of nonempty subsets $S_X \subseteq X$ and $S_Y \subseteq Y$ of vertices such that the induced directed graph $H[S_X \cup S_Y]$ has no cycle and for any two vertices $v, u \in S_*$ ($S_* = S_X$ or S_Y) it holds $N^+(v) \setminus (S_X \cup S_Y) = N^+(u) \setminus (S_X \cup S_Y)$ and $N^-(v) \setminus (S_X \cup S_Y) = N^-(u) \setminus (S_X \cup S_Y)$. A bimodule $B = (S_X, S_Y)$ is *maximal* if it cannot be extended by adding vertices into S_X or S_Y.

Lemma 1. *There is at most one maximal bimodule that contains any pair of non-similar vertices a and b on the same side of a bipartite tournament $H = (X \cup Y, A)$ (i.e., either $\{a, b\} \subseteq X$ or $\{a, b\} \subseteq Y$).*

Proof. With out loss of generality, we assume that $\{a, b\} \subseteq X$. Suppose to the contrary that there are two different maximal bimodules $B_1 = (X_1, Y_1)$ and $B_2 = (X_2, Y_2)$ such that $\{a, b\} \subseteq X_1 \cap X_2$.

We claim that there exist a vertex $c \in Y_1 \cap Y_2$ such that a, b and c form a length-3 directed path acb or bca in H (without loss of generality, we assume the path is acb). If there is no such a vertex $c \in Y_1 \cap Y_2$, then for each vertex $y \in Y$ the two arcs between $\{y\}$ and $\{a, b\}$ have the same direction and then a and b are similar vertices, a contradiction.

We also claim that there is a cycle in $H[B_1 \cup B_2]$. For each vertex $x_0 \in X \setminus X_1$ (resp., $y_0 \in Y \setminus Y_1$), all the arcs between x_0 and Y_1 (resp., between y_0 and X_1) have the same direction as (x_0, c) (resp. (y_0, a)), since $B_1 = (X_1, Y_1)$ is a bimodule. Then for each vertex $x_0 \in X \setminus (X_1 \cup X_2)$ (resp. $y_0 \in Y \setminus (Y_1 \cup Y_2)$), all the arcs between x_0 and Y_1 (resp. between y_0 and X_1) have the same direction as (x_0, c) (resp. (y_0, a)). By symmetry, for each vertex $x_0 \in X \setminus (X_2 \cup X_1)$ (resp. $y_0 \in Y \setminus (Y_2 \cup Y_1)$), all the arcs between x_0 and Y_2 (resp. between y_0 and X_2) have the same direction as (x_0, c) (resp. (y_0, a)). Therefore, for each vertex $x_0 \in X \setminus (X_1 \cup X_2)$ (resp. $y_0 \in Y \setminus (Y_1 \cup Y_2)$), all the arcs between x_0 and $Y_1 \cup Y_2$ (resp. between y_0 and $X_1 \cup X_2$) have the same direction. Since B_1 and B_2 cannot be joined into one big bimodule, by the defintion of bimodules, we know that there is a cycle in $H[B_1 \cup B_2]$.

It is easy to verify that a tournament contains a cycle iff it contains a length-4 cycle. Let C denote a length-4 cycle in $H[B_1 \cup B_2]$. Clearly, cycle C cannot be completely contained in B_1 or B_2. Moreover, $C \cap (B_1 \cap B_2) = \emptyset$, since, otherwise, one of B_1 and B_2 would contain three vertices of C and by the definition of bimodules the two arcs between the remaining vertex of C and these three vertices should have the same direction, contradicting the fact that C is a cycle. Therefore, $|C \cap B_1| = 2$, $|C \cap B_2| = 2$, and $C \cap (B_1 \cap B_2) = \emptyset$. Let u, v, w, x be the vertices of C with $\{(u, v), (v, w), (w, x), (x, u)\} \subseteq A$, $\{u, v\} = C \cap B_1$, and $\{w, x\} = C \cap B_2$.

We will try to find some contradiction based the cycle C and the path acb. If $u \in X$ (now $\{u, w, a, c\} \subseteq X$ and $\{v, x, b\} \subseteq Y$), then, since B_1 is a bimodule, the arcs between $\{a, c\}$ and x should have the same direction as (x, u) and the arc between b and w should have the same direction as (v, w). This implies a cycle formed by a, b, w and x which is completely contained in B_2, contradicting the fact that B_2 is a bimodule. The other case of $u \in Y$ will lead to the same contradiction, which completes the proof. ∎

Lemma 2. *If there is a bimodule containing a pair of non-similar vertices a and b on the same side of a bipartite tournament H, then the maximal bimodule B containing a and b can be found in polynomial time.*

Proof. Assume that $\{a, b\} \subseteq X$. We construct the maximal bimodule in a greedy manner. Initially let $X' = \{a, b\}$ and $Y' = \emptyset$. We repeat the following two steps until $H[X' \cup Y']$ becomes cyclic or $X' \cup Y'$ cannot be changed. Step 1: (1) If there are vertices in Y that have arcs with different directions to X', then add these vertices to Y'; (2) Switch the roles of X and Y and of X' and Y' and go to (1). Step 2: If $H[X' \cup Y']$ becomes cyclic, then there is no bimodule containing a and b; otherwise, we iterate over all vertices in $X \setminus X'$ and $Y \setminus Y'$ and check for each of them whether it can be added to X' and Y' without destroying the properties of bimodules. Finally, we output the resulting X' and Y'.

The soundness of the above procedure is based on the observation that, if there is a bimodule containing a and b, then all vertices satisfying Condition (1) must be in the bimodule. Then, by Lemma 1, the above procedure correctly constructs the maximal bimodule containing a and b. The polynomial running time is easy to verify. ∎

3.2 k-Central and k-Redundant Vertices

Let $B = (S_X, S_Y)$ be a bimodule of a bipartite tournament $H = (X \cup Y, A)$. A vertex u in S_* ($* \in \{X, Y\}$) is *k-central* if there are two topological sorts T and T' of $H[S_X \cup S_Y]$ (T and T' could be identical) such that at least $k+1$ vertices in S_* are on the left of u in T and at least $k+1$ vertices in S_* are on the right of u in T'. A vertex u is *k-redundant* if it is $(k+1)$-central and S_* contains another vertex u' which is similar to u. It is easy to see that if S_* contains a module M of size at least $k+2$, then every vertex in M is k-central and k-redundant.

Lemma 3. *A k-redundant vertex in a bimodule can be found in polynomial time if it exists.*

Proof. Let T be an arbitrary topological sort of the bimodule B. By the definition of topological sorts, we can see that we can switch the seats of two vertices in T to get another topological sort iff the two vertices are similar. To check whether a vertex u in B is k-redundant or not, we only need to do this: Move all vertices that are similar to u to the left (resp. right) of u in T and check if there are at least $k+1$ vertices on the left (resp. right) of u. ∎

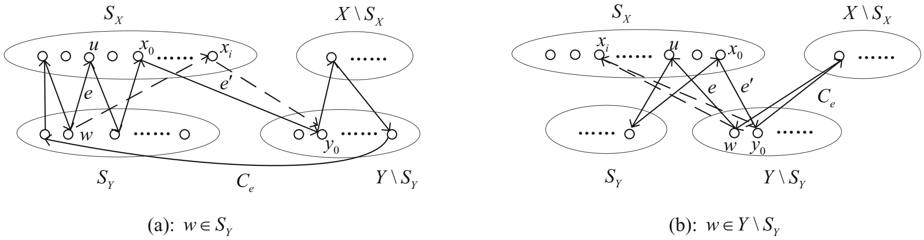
Lemma 4. *Let u be a k-redundant vertex in a bimodule B of a bipartite tournament H. Every vertex u' in B that is similar to u is a k-central vertex in the bimodule $B' := B \setminus \{u\}$ of $H' := H \setminus \{u\}$.*

Proof. It is easy to see that B' is a bimodule of H'. Note that we can exchange the positions of u and u' in every topological sort T of B to get another topological sort T'. By the symmetry, we know that u' is also k-redundant (and hence $(k+1)$-central) in B. After removing u from B, vertex u' is clearly k-central in B'. ∎

The following two lemmas are crucial for the correctness of our kernelization algorithm.

Lemma 5. *Let $H = (X \cup Y, A)$ be a bipartite tournament and u be a k-central vertex in a bimodule $B = (S_X, S_Y)$ of H. No minimal feedback arc set of size at most k of H contains arcs incident on u.*

Proof. Without loss of generality, we assume that the k-central vertex u is in S_X. By the definition of k-central vertices, we know that $|S_X| \geq k+2$. Note that in this lemma, we do not require that S_X contains two non-similar vertices, i.e., it is possible that all vertices in S_X are in a same module. Let F be a minimal



(a): $w \in S_Y$

(b): $w \in Y \setminus S_Y$

Fig. 1. Arc $e' = (x_0, y_0)$ is in $A(S_X, Y \setminus S_Y)$

feedback arc set of size at most k. Assume to the contrary that F contains an arc e incident on u. We will show a contradiction.

Since F is minimal, there is a cycle C_e in H containing only one arc e in F. Since $H[S_X \cup S_Y]$ has no cycle, C_e contains some arc e' in $A(S_X \cup S_Y, X \cup Y \setminus (S_X \cup S_Y)) = A(S_X, Y \setminus S_Y) \cup A(S_Y, X \setminus S_X)$.

First of all, we show that e' cannot be in $A(S_X, Y \setminus S_Y)$. Assume to the contrary that C_e contains an arc $e' \in A(S_X, Y \setminus S_Y)$. We will show a contradiction that $|F| > k$. Assume $e' = (x_0, y_0)$ (resp. $e' = (y_0, x_0)$), where $x_0 \in S_X$ and $y_0 \in Y \setminus S_Y$. Let w be the vertex in C_e such that there is an arc (w, u) (resp. (u, w)). Then the path P from y_0 to w (resp. from w to y_0) in C_e does not contain any arc in F. See Figure 1 for the reference. Next we show that there are at least $k+1$ different vertices $\{x_i \mid 1 \leq i \leq k+1\}$ in S_X such that $P \cup A(x_i, \{w, y_0\})$ induces a graph C_{x_i} that is a cycle or two arc-disjoint cycles (the later case happens only when x_i is a vertex in P). Then H needs at least $k+1$ different arcs to intersect all the cycles in the $k+1$ graphs C_{x_i} (note that each pair of the cycles only share path P where all arcs are not in F), which is contradicting the fact that $|F| \leq k$. To find $\{x_i \mid 1 \leq i \leq k+1\}$, we consider two cases. Case 1: $w \in S_Y$ (see Figure 1(a)). There are at least $k+1$ different vertices $\{x_i \mid 1 \leq i \leq k+1\}$ in S_X such that all the arcs between $\{x_i \mid 1 \leq i \leq k+1\} \cup \{u\}$ and $\{w\}$ have the same direction by the definition of k-central vertices. All the arcs between $\{x_i \mid 1 \leq i \leq k+1\} \cup \{x_0\}$ and $\{y_0\}$ also have the same direction by the definition of bimodules. If x_i is not in P, then C_{x_i} is a cycle. If x_i is in P, then C_{x_i} is a graph of two arc-disjoint cycles (sharing a common vertex x_i). Case 2: $w \in Y \setminus S_Y$ (see Figure 1(b)). For any vertex $x \in S_X$ the arcs between $\{x, u\}$ and $\{w\}$ (resp. between $\{x, u\}$ and $\{y_0\}$) are of the same direction by the definition of bimodules. Then x_i's can be any $k+1$ different vertices in S_X.

By the fact that $C_e \cap A(S_x, Y \setminus S_Y) = \emptyset$, cycle C_e contains at least two arcs in $A(S_Y, X \setminus S_X)$. One is from S_Y to $X \setminus S_X$ and one is from $X \setminus S_X$ to S_Y (see Figure 2). We assume that they are (x_1, y_1) and (y_2, x_2), where $x_i \in X \setminus S_X$ and $y_i \in S_Y$ for $i = 1, 2$. Note that by the definition of bimodules, $x_1 \neq x_2$. With the same reason, we can safely assume that the path from x_2 to x_1 in C_e does not contain any vertex in $S_X \cup S_Y$ (such an arc pair (x_1, y_1) and (y_2, x_2) always exists). Let $w \neq y_2$ be the other vertex adjacent to x_2 in C_e (i.e. there is an arc (x_2, w)). By the selection of the arc pair, vertex w is clearly not in S_Y.

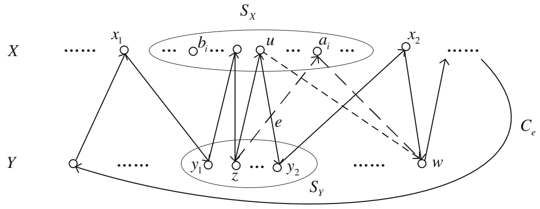


Fig. 2. Cycle C_e contains two arcs (x_1, y_1) and (y_2, x_2) in $A(S_Y, X \setminus S_X)$

Now consider the arc between u and w. In the case that $(u, w) \in A$, all vertices in S_X have arcs to w. Let z be the vertex on C_e with (z, u) and P' be the path in C_e from w over x_1 to z. Moreover, by the case discussed above, we have $z \in S_Y$. Since u is k-central, there are at least $k+1$ vertices $\{a_i \mid 1 \leq i \leq k+1\}$ with (z, a_i) for all i's. For each i, the arcs (z, a_i) and (a_i, w) together with the path P' form one or two arc-disjoint cycles, depending on whether a_i is on P'. Therefore, F must contain at least one of (a_i, w) and (z, a_i) for each i, contradicting the fact that $|F| \leq k$. The case $(w, u) \in A$ can be argued in the same way. We consider then the vertex z' on C_e with (u, z'). There are $k + 1$ vertices b_i's with (b_i, z') and (w, b_i). Then the arcs (b_i, z') and (w, b_i) together with path P'' (the path in C_e from z' over x_2 to w) form one or two arc-disjoint cycles. Therefore, we know that if $|F| \leq k$ then F cannot contain an arc incident on u. ∎

Lemma 6. *Let H be a bipartite tournament and u be a k-redundant vertex in a bimodule B of H. Graph H has a feedback arc set of size at most k if and only if H' has a feedback arc set of size at most k, where $H' = H \setminus \{u\}$ is the graph obtained by deleting u from H.*

Proof. Clearly, every feedback arc set of H is a feedback arc set of H', since H' is a subgraph of H. Next, we only need to prove that if H' has a feedback arc set of size at most k then H also has one.

Assume that F is a minimal feedback arc set in H' with $|F| \leq k$. We show that F is also a feedback arc set in H. Let u' be a vertex $u' \neq u$ in the same module of u (u' exists since u is k-redundant). Then u' is k-central in H' (by Lemma 4) and no arc incident on u' is in F (by Lemma 5). Assume to the contrary that F is not a feedback arc set of H. Then there is a cycle C in H that does not contain any arc in F but contains vertex u. If u' is not in C, then we can get another cycle C' by replacing u with u' in C. Note that C' does not contain any arc in F, since no arc incident on u' is in F. Then C' also is a cycle in H' that does not contain any arc in F, which is a contradicting the fact that F is a feedback arc set of H'. Next, we assume that u' is in C. Note that u' and u cannot have a common neighbor in C, since u' and u are similar. Let $w \neq u'$ be the vertex on C with (w, u). We can then replace the path in C between w and u' that contains u by the single arc between w and u' to get another cycle C'. Cycle C' is a cycle in H' containing no arc in F, also a contradiction. Therefore, we know that F is also a feedback arc set of H, which completes the proof. ∎

4 Data Reduction Rules

Now we are ready to introduce our reduction rules that can be applied in polynomial time on an instance of k-FASBT to get an equivalent instance with "smaller" size.

Rule 1. *Delete any vertex from the bipartite tournament if the vertex in not contained in any cycle.*

Rule 2. *If $k < 0$, report that there is no feedback arc set of size at most k.*

Rule 3. *If there are an arc e and at least $k+1$ 4-cycles such that each pair of the 4-cycles have only one common arc e, then reverse e and reduce k by one, i.e., return the instance $(H\{e\}, k-1)$.*

Rule 4. *If there is a module M of size more than $k+1$, then remove vertices from M until $|M| = k+1$.*

Rule 5. *For every pair of non-similar vertices on the same side of the bipartite tournament, compute the maximal bimodule B containing them, if such one exists, and iteratively remove a k-redundant vertex from B until no such kind of vertices exist.*

We say that a reduction rule is *correct*, if, whenever it transforms an instance (H, k) to (H', k'), graph H has a solution of size at most k iff H' has a solution of size at most k', and $k \geq k'$. All the above reduction rules can be applied in polynomial time. The correctness of the first two reduction rules is obvious, the correctness of Rule 3 is based on Proposition 1, and the correctness of Rules 4 and 5 is based on Lemma 6 (note that any vertex in a module M of size at least $k+2$ is a k-redundant vertex in some bimodule containing M). In fact, Rule 4 is introduced in [13] to get a cubic vertex kernel for k-FASBT.

We say a bipartite tournament is *reduced*, if it cannot be changed by applying the above rules.

Lemma 7. *For every pair of non-similar vertices on the same side in a reduced bipartite tournament $H = (X \cup Y, A)$, if there is maximal bimodule B containing them, then B has at most $6k+2$ vertices.*

Proof. Assume that there is a maximal bimodule $B = (S_X, S_Y)$ that contains the three non-similar vertices and has more than $6k+2$ vertices. We show that at least one reduction rule can be applied to the bipartite tournament. Let T be a topological sort of B. One of S_X and S_Y, say S_X, contains at least $3k+2$ vertices. Considering the ordering T_X that is the restriction of T to S_X, we use Z to denote the set of the S_X-vertices of the i-th positions with $k+2 \leq i \leq 2k+1$ from the left to the right in T_X. If there was one vertex z in Z whose left or right neighbor in T is also in S_X, then this neighbor should be in the same module as z. This would imply that vertex z is a k-redundant vertex and Rule 5 can be applied. Thus, both neighbors of z in T are vertices in S_Y for all $z \in Z$ and $|S_Y| \geq k+1$ (notice that $|Z| = k$). Since Rule 1 cannot be applied, all vertices in B are contained in some cycles, in particular, in some length-4 cycles, in H. Let z be a vertex in Z and C be a length-4 cycle in H containing z, i.e., $C = \{z, a, b, c\}$ with (z, a), (a, b), (b, c), and (c, z). Moreover, we use $l(z)$ and $r(z)$ to denote the left and right neighbor of z in T, respectively. Clearly, $l(z), r(z) \in S_Y$. By definition of bimodules, $|C \cap B| \leq 2$. If $|C \cap B| = 2$, then either $\{z, a\} \subseteq B$ or $\{z, c\} \subseteq B$. If the former case applies, then we have the arc $(r(z), b)$ and a cycle of $z, r(z), b, c$. Moreover, by the definition of bimodules, we have a length-4 cycle $z', r(z'), b, c$ for every $z' \in Z$. Since $|S_Y| \geq k+1$, we have then more than k cycles sharing only the arc (b, c) and Rule 3 would then apply. If $|C \cap B| = 1$, then consider the arcs between S_Y and b. If they are directed from S_Y to b, then we have for each $z \in Z$ a length-4 cycle containing arcs $(z, r(z))$ and (b, c); else, the cycle contains $l(z)$ and (a, b). This implies again that there are more than k length-4 cycles pairwise intersecting only in one arc and Rule 3 would apply. ∎

Now we have all tools to prove the kernel.

Theorem 1. *k-FASBT admits a problem kernel with $12k^2 - 2$ vertices.*

Proof. If suffices to prove that, if a reduced bipartite tournament H has a feedback arc set of size at most k, then H has at most $12k^2 - 2$ vertices.

Let F be a feedback arc set of size at most k in H and T be a topological sort of the graph after reversing the arcs in F, where all arcs go from left to right. For convenience, we say that vertices that appear in some arc in F are *affected* vertices and the other vertices are *unaffected* vertices. Affected vertices separate T into several sections which are groups of unaffected vertices (called *spans*). There are at most $2k$ affected vertices and at most $2k - 1$ spans (the first and last vertices in T should be affected vertices, since otherwise Rule 1 would apply). If a span contains vertices from at most two modules, then this span contains at most $2k + 2$ vertices, due to Rule 4. If the span contains vertices from at least three modules, then choose arbitrary two vertices a and b from two modules on the same side. We claim that H contains a maximal bimodule B containing a and b and the vertices in the span are contained in B. The reason for this claim is as follows: The vertices of the span induce clearly an acyclic graph and satisfy the neighborhood condition of bimodules. By Lemma 1, there is only one maximal bimodule containing them. Lemma 7 implies that this span contains at most $6k+2$ vertices. Therefore, graph H contains at most $(6k+2)(2k-1)+2k = 12k^2 - 2$ vertices. ∎

5 Conclusion

By introducing the new concept "bimodule," we have improved the size of the kernel for k-FASBT from $O(k^3)$ vertices to $O(k^2)$ vertices. It remains open whether this bound can be further lowered to $O(k)$. To this end, a more thoroughful study of the bimodule structure could be a promising starting point.

Acknowledgements. The authors thank all the anonymous reviewers for their comments to improve the presentation of Lemma 1 and some other places in the paper.

References

1. Abu-Khzam, F.N.: A kernelization algorithm for d-hitting set. Journal of Computer and System Sciences 76(7), 524–531 (2010)
2. Alon, N.: Ranking tournaments. SIAM J. Discrete Math. 20(1), 137–142 (2006)
3. Alon, N., Lokshtanov, D., Saurabh, S.: Fast FAST. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 49–58. Springer, Heidelberg (2009)
4. Cai, M., Deng, M., Zang, W.: A min-max theorem on feedback vertex sets. Math. Oper. Res. 27(2), 361–371 (2002)
5. Bessy, S., Fomin, F.V., Gaspers, S., Paul, C., Perez, A., Saurabh, S., Thomassé, S.: Kernels for feedback arc set in tournaments. J. Comput. Syst. Sci. 77(6), 1071–1078 (2009)
6. Charbit, P., Thomassé, S., Yeo, A.: The minimum feedback arc set problem is NP-hard for tournaments. Combin. Probab. Comput. 16(1), 1–4 (2007)
7. Dom, M., Guo, J., Hüffner, F., Niedermeier, R., Truß, A.: Fixed-parameter tractability results for feedback set problems in tournaments. J. Discrete Algorithms 8(1), 76–86 (2010)
8. Guo, J., Hüffner, F., Moser, H.: Feedback arc set in bipartite tournaments is NP-complete. Inf. Process. Lett. 102(2-3), 62–65 (2007)
9. Hsiao, S.-Y.: Fixed-Parameter Complexity of Feedback Vertex Set in Bipartite Tournaments. In: Asano, T., Nakano, S.-i., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 344–353. Springer, Heidelberg (2011)
10. Karpinski, M., Schudy, W.: Faster Algorithms for Feedback Arc Set Tournament, Kemeny Rank Aggregation and Betweenness Tournament. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part I. LNCS, vol. 6506, pp. 3–14. Springer, Heidelberg (2010)
11. Kemeny, J.: Mathematics without numbers. Daedalus 88, 571–591 (1959)
12. Kemeny, J., Snell, J.: Mathematical models in the social sciences. Blaisdell (1962)
13. Hsiao, S.-Y.: Fixed-Parameter Complexity of Feedback Vertex Set in Bipartite Tournaments. In: Asano, T., Nakano, S.-i., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 344–353. Springer, Heidelberg (2011)
14. Paul, C., Perez, A., Thomassé, S.: Conflict Packing Yields Linear Vertex-Kernels for k-FAST, k-DENSE RTI and a Related Problem. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 497–507. Springer, Heidelberg (2011)
15. Sanghvi, B., Koul, N., Honavar, V.: Identifying and Eliminating Inconsistencies in Mappings across Hierarchical Ontologies. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6427, pp. 999–1008. Springer, Heidelberg (2010)
16. Speckenmeyer, E.: On Feedback Problems in Digraphs. In: Nagl, M. (ed.) WG 1989. LNCS, vol. 411, pp. 218–231. Springer, Heidelberg (1990)

17. Sasatte, P.: Improved FPT algorithm for feedback vertex set problem in bipartite tournament. Information Processing Letters 105(3), 79–82 (2008)
18. Wahlström, M.: Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems. PhD thesis, Linköpings universitet (2007)
19. Raman, V., Saurabh, S.: Parameterized algorithms for feedback set problems and their duals in tournaments. Theor. Comput. Sci. 351(3), 446–458 (2006)

Author Index