Edward A. Hirsch
Juhani Karhumäki
Arto Lepistö
Michail Prilutskii (Eds.)

LNCS 7353

Computer Science – Theory and Applications

7th International Computer Science Symposium in Russia, CSR 2012
Nizhny Novgorod, Russia, July 2012, Proceedings

Springer

Lecture Notes in Computer Science 7353

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

Edward A. Hirsch Juhani Karhumäki
Arto Lepistö Michail Prilutskii (Eds.)

Computer Science – Theory and Applications

7th International Computer Science Symposium in Russia, CSR 2012
Nizhny Novgorod, Russia, July 3-7, 2012
Proceedings

Springer

Volume Editors

Edward A. Hirsch
Steklov Institute of Mathematics at St.Petersburg
191023 St. Petersburg, Russia
E-mail: hirsch@pdmi.ras.ru

Juhani Karhumäki
Arto Lepistö
University of Turku, 20014 Turku, Finland
E-mail: {karhumak, alepisto}@utu.fi

Michail Prilutskii
Lobachevsky State University of Nizhny Novgorod
603950 Nizhny Novgorod, Russia
E-mail: pril@iani.unn.ru

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-30641-9 e-ISBN 978-3-642-30642-6
DOI 10.1007/978-3-642-30642-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012938206

CR Subject Classification (1998): F.2, F.3, E.3, G.2, F.1, F.4

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 7th International Computer Science Symposium in Russia (CSR 2012) was held during July 3–7, 2012, in Nizhny Novgorod, Russia, hosted by the Lobachevsky State University. It was the seventh event in the series of regular international meetings, following CSR 2006 in St. Petersburg, CSR 2007 in Ekaterinburg, CSR 2008 in Moscow, CSR 2009 in Novosibirsk, CSR 2010 in Kazan and CSR 2011 in St. Petersburg. CSR 2012 was one of the events of the Alan Turing Year 2012. It took place under the auspices of European Association of Theoretial Computer Science (EATCS).

The opening lecture was given by Vijay Vazirani and the special Turing lecture by Yuri Matiyasevich. Other invited plenary lecturers were Lev Beklemishev, Mikołaj Bojańczyk, Julien Cassaigne, Piotr Indyk, Jaroslav Nešetřil and Pavel Pevzner.

This volume contains the accepted papers and abstracts of the invited papers. The scope of the topics of the symposium was broad and covered substantial parts of theoretical computer science and its applications. We received 66 submissions. Of these, the international Program Committee selected 28 for presentation at the conference, using the EasyChair system. A special issue of *Theory of Computing Systems* consisting of selected papers will be published.

As usual, Yandex provided the Best Paper Awards; the recipients of these were selected by the Program Committee:

- Best paper award: C. Kapoutsis and G. Pighizzini, "Two-Way Automata Characterizations of L/poly Versus NL"
- Best student paper: E. Demenkov, "A Lower Bound on Circuit Complexity of Vector Function in U_2"

The following satellite events were co-located with CSR 2012:

- The Third Workshop on Program Semantics, Specification and Verification: Theory and Applications (PSSV 2012)
- Workshop on Current Trends in Cryptology (CTCrypt 2012)

We are grateful to our sponsors:

- Dynasty Foundation
- Finnish Academy of Science and Letters, mathematics funding
- Microsoft Research
- Russian Foundation for Basic Research
- Yandex

Last but not least, we thank Springer for the smooth cooperation, and the local organizers, in particular, Roman Strongin (President of State University of Nizhni Novgorod) for their great help and support.

April 2012

Edward Hirsch
Juhani Karhumäki
Arto Lepistö
Michail Prilutskii

Organization

CSR 2012 intended to reflect the broad scope of international cooperation in computer science. It was the seventh conference in a series of regular events started with CSR 2006 in St. Petersburg. The topics covered vary from year to year, but in general try to cover as much of the field of contemporary computer science as possible.

CSR 2012 was included in schedule of the Alan Turing Year events.

Conference Chair

Michail Prilutskii	Lobachevsky State University of Nizhni Novgorod, Russia

Program Chair

Juhani Karhumäki	University of Turku, Finland

Program Committee

Lev Afraimovich	Lobachevsky State University of Nizhni Novgorod, Russia
Susanne Albers	HU Berlin, Germany
Andris Ambainis	University of Latvia, Latvia
Alberto Bertoni	University of Milan, Italy
Bruno Durand	Université Montpellier 2, France
Edward A. Hirsch	Steklov Institute of Mathematics at St. Petersburg, Russia
Juraj Hromkovic	ETH Zürich, Switzerland
Kazuo Iwama	Kyoto University, Japan
Juhani Karhumäki	University of Turku, Finland
Markus Lohrey	University of Leipzig, Germany
Ernst Mayr	TU München, Germany
Ilya Mironov	Microsoft Research Silicon Valley, USA
Anca Muscholl	Université Bordeaux, France
Alexander Okhotin	University of Turku, Finland
Alexander Razborov	University of Chicago, USA
Wojciech Rytter	Warsaw University, Poland
Jirí Sgall	Charles University, Czech Republic
Alexander Shen	Université Montpellier 2, France
Arseny Shur	Ural State University, Russia
Wolfgang Thomas	RWTH Aachen University, Germany
Nikolai Vereshchagin	Moscow State University, Russia

Gerhard Woeginger	Eindhoven University of Technology, The Netherlands
Mikhail Vyalyi	Dorodnicyn Computing Centre of RAS, Russia

Steering Committee

Volker Diekert	University of Stuttgart, Germany
Anna Frid	Sobolev Institute of Mathematics, Russia
Edward A. Hirsch	Steklov Institute of Mathematics at St. Petersburg, Russia
Juhani Karhumäki	University of Turku, Finland
Mikhail Volkov	Ural State University, Russia

Organizing Committee

Michail Prilutskii	Lobachevsky State University of Nizhni Novgorod, Russia
Victor Gergel	Lobachevsky State University of Nizhni Novgorod, Russia
Vladimir Shvetsov	Lobachevsky State University of Nizhni Novgorod, Russia
Lev Afraimovich	Lobachevsky State University of Nizhni Novgorod, Russia
Alexander Gorylev	Lobachevsky State University of Nizhni Novgorod, Russia
Vadim Saigin	Lobachevsky State University of Nizhni Novgorod, Russia
Yana Safonova	Lobachevsky State University of Nizhni Novgorod, Russia

Referees

S. Aguzzoli	G. Calinescu	A. Grigoriev
A. Antoniadis	A. Carayol	E. Grädel
L. Babai	M. Chrobak	P. Guillon
V. Barany	W. Czerwinski	V. Gusev
N. Benes	B. Das	L. Hellerstein
C. Blundo	O. Dubois	J. Hitchcock
M. Bodirsky	R. Ehlers	D. Itsykson
H.J. Boeckenhauer	M. Fellows	E. Jeandel
P. Bonizzoni	T. Fernique	A. Jèz
D. Bruschi	E. Formenti	J. Johannsen
A. Bulatov	P. Gawrychowski	J. Kari
J.Y. Cai	K. Georgatos	A. Kartzow

D. Kirsten
D. Komm
G. Kortsarz
S. Krug
J. Krugel
P. Krysta
A. Kulikov
J. Lahtonen
M. Lange
M. Lauria
T. Lehtinen
P. Lenzner
L. Lyaudet
C. Löding
D. Manlove
C. Mereghetti
I. Mezhirov
S. Miguet
I. Mironov
S. Miyazaki
H. Morizumi
D. Musatov
T. Mömke
S. Nikolenko
J. Olschewski
B.M. Paola
D. Pardubska
P. Parys
I. Petre
G. Pighizzini
C. Pinkau
V. Podolskii
Y. Pritykin
M. Raskin
K. Reinhardt
A. Reuss
H. Rivano
A. Roman
A. Romashchenko
A. Salomaa
K. Salomaa
H. Schnoor
C. Shah
V. Shpilrain
S. Smyczynski
F. Spieksma
A. Sprock
R. Stefanec
M. Steinova
P. Tesson
D. Thilikos
W. Tyczynski
H. Täubig
T. Walen
R.F.C. Walters
J. Weihmann
D. West
A. Wolff
X. Wu
M. Zimand

Sponsoring Institutions

Dynasty Foundation, Russia
Finnish Academy of Science and Letters, Finland
Microsoft Research, USA
Russian Foundation for Basic Research, Russia
Yandex, Russia

Turing Talk

Alan Turing and Number Theory

Yuri V. Matiyasevich

Steklov Institute of Mathematics (POMI)
St.Petersburg, Russia 191023
bengio@google.com

Beside well-known revolutionary contributions, Alan Turing had a number of significant results in "traditional" mathematics. In particular he was very much interested in the famous Riemann Hypothesis. This hypothesis, stated by Berhard Riemann in 1859 and included by David Hilbert in his 8th problem in 1990, still remains open, being now one of the Millennium Problems. The Riemann Hypothesis predicts positions of zeros of so called zeta function, and Alan Turing developed a rigorous method for verifying the Hypothesis for the initial zeros. He also invented a machine for calculating the values of the zeta function. In contrast to celebrated imaginable Turing machines, Turing started to implement this machine but never finished because of the War.

Plenary Invited Talks

Challenges in Comparative Genomics: From Biological Problems to Combinatorial Algorithms (and back)*

Max A. Alekseyev[1,3] and Pavel Pevzner[2,3]

[1] University of South Carolina, Columbia, SC, U.S.A.
[2] Department of Computer Science and Engineering
University of California in San Diego, La Jolla, CA, U.S.A.
[3] Academic University, St. Petersburg, Russia

Recent large-scale sequencing projects fueled the comparative genomics studies and heightened the need for algorithms to extract valuable information about genetic and phylogenomic variations. Since the most dramatic genomic changes are caused by genome rearrangement events (which shuffle genomic material), it becomes extremely important to understand their mechanism and reconstruct the sequence of such events (evolutionary history) between genomes of interest.

In this expository talk I shall describe several controversial and hotly debated topics in evolutionary biology (chromosome breakage models, mammalian phylogenomics, prediction of future rearrangements) and formulate related combinatorial challenges (rearrangement and breakpoint re-use analysis, ancestral genomes reconstruction problem). I shall further present recent theoretic and algorithmic advances in addressing these challenges and their biological implications.

* This work was partially supported by the Government of the Russian Federation (grant 11.G34.31.0018).

DKAL: A Distributed Knowledge Authorization Language and Its Logic of Information

Lev Beklemishev

Steklov Institute of Mathematics, Moscow, Russia

This talk is a report on the DKAL project developed at Microsoft Research by Yuri Gurevich and his collaborators (Itay Neeman, Michal Moskal, Andreas Blass, Guido di Caso, and the others). I will survey some of the main features of DKAL and discuss the underlying information logics.

With the advent of cloud computing, the role of formal policies grows. The personnel of brick-and-mortar businesses often exercise their judgments; all that should be replaced with formal policies when businesses move to the cloud. The logic-based policy language DKAL (Distributed Knowledge Authorization Language) [4, 5, 3] was developed with such applications in mind. The feature that distinguishes DKAL from most preceding logic-based policy languages is that it is explicitly geared toward federated scenarios (with no central authority) where trust may be in short supply.

The world of DKAL consists of communicating principals computing their own knowledge in their own states. They communicate *infons*, pieces of information, and reason in terms of infons. In [5], the original developers of DKAL distilled the basic features of the logic of infons and introduced infon logic **qI** that is an extension of the $\{\rightarrow, \wedge\}$ fragment **I** of intuitionistic logic with quotation modalities $p\,\mathtt{said}\,\phi$ and $p\,\mathtt{implied}\,\phi$. In addition they isolated a *primal fragment* **qP** of **qI** which is very efficient and yet sufficiently expressive for many purposes. In the case of bounded quotation depth, the derivation problem for **qP** is solvable in linear time. In particular, the quotation-free fragment **P** of **qP** is linear time in that sense.

The continuing development of DKAL (whose current implementation is found at [1]) requires further investigation of the logic of infons. In [2], we extend the four logics of [5] with one or both of disjunction and negation, and we determine the complexities of the extended logics. We provide a semantics for the extension $\mathbf{P}[\vee]$ of **P** that we call *quasi-boolean*. This allows us to give efficient mutual translations between $\mathbf{P}[\vee]$ and classical propositional logic as well as an embedding of the appropriate classical modal logic into $\mathbf{qP}[\vee]$. On the proof-theoretic side we develop cut-free Gentzen-style sequent calculi for the extensions of primal logic **P** with some or all of disjunction, negation and quotations.

References

1. DKAL at CodePlex: `http://dkal.codeplex.com/`
2. Beklemishev, L., Gurevich, Y.: Propositional primal logic with disjunction. Technical Report MSR-TR-2011-35, Microsoft Research (March 2011); To appear in Journal on Logic and Computation
3. Blass, A., Gurevich, Y., Moskal, M., Neeman, I.: Evidential authorization. In: Nanz, S. (ed.) The Future of Software Engineering, pp. 77–99. Springer (2011)
4. Gurevich, Y., Neeman, I.: DKAL: Distributed-Knowledge Authorization Language. In: Proc. of CSF 2008, pp. 149–162. IEEE Computer Society (2008)
5. Gurevich, Y., Neeman, I.: DKAL 2 — A Simplified and Improved Authorization Language. Technical Report MSR-TR-2009-11, Microsoft Research (February 2009)

Infinite Sets That Are Finite Up to Permutations

Mikołaj Bojańczyk*

University of Warsaw

Fraenkel-Mostowski sets are a variant of set theory, where sets can contain atoms. The existence of atoms is postulated as an axiom. The key role in the theory of Fraenkel-Mostowski sets is played by permutations of atoms. For instance, if a, b, c, d are atoms, then the sets

$$\{a, \{a, b, c\}, \{a, c\}\} \qquad \{b, \{b, c, d\}, \{b, d\}\}$$

are equal up to permutation of atoms.

Fraenkel-Mostowski sets were rediscovered for the computer science community, by Gabbay and Pitts [3]. It turns out that atoms are a good way of describing variable names in programs or logical formulas, and the permutations of atoms are a good way of describing renaming of variables. Fraenkel-Mostowski sets are now widely studied in the semantics community, under the name of nominal sets (the name is so chosen because atoms describe variables names).

This talk is about a new use application of these sets in computer science. (We use the name nominal sets.) The new application has its roots in database theory, but it touches other fields, such as verification or automata theory. The motivation in database theory is that atoms can be used as an abstraction for data values, which can appear in a relational database or in an XML document. Like in nominal sets, there are infinitely many possible atoms (or data values), but properties of databases (relational or XML) are closed under permutations of atoms. Atoms can also be used to model sources of infinite data in other applications, such as software verification, where an atom can represent a pointer or the contents of an array cell.

Nominal sets are a good abstraction for infinite systems because they have a different, more relaxed, notion of finiteness. A nominal sets is considered finite if it is finite up to permutation of atoms. (We call such a set orbit-finite.) For instance, the set of atoms is orbit-finite, because every atom can be mapped to every other atom by a permutation. Likewise, the set of pairs of atoms has two elements up to permutation, namely (a, a) and (a, b) for $a \neq b$. Another example is the set of λ-terms which represents the identity, up to α-conversion:

$$\{\lambda a.a : a \text{ is an atom}\}.$$

Yet another example concerns automata with registers for storing atoms [4]: up to permutation, there are finitely many configurations of an automaton which has four control states and three registers which can store atoms.

* Author supported by ERC Starting Grant "Sosna".

The language of nominal sets is so robust that one can meaningfully restate all of the results in a textbook on automata theory, replacing sets by nominal sets and finite sets by orbit-finite sets, see [2] for examples. Some of the restated theorems are true, some are not. Results that work in the nominal setting include the Myhill-Nerode theorem, or the equivalence of pushdown automata with context free grammars. Results that fail in the nominal setting include all results which depend on the subset construction, such as determinization of finite automata, or equivalence of two-way and one-way finite automata.

Perhaps most importantly, the theory of computability still works in the nominal setting. More specifically, one can design a programming language which manipulates orbit-finite nominal sets, just like other programming languages manipulate lists or trees [1]. (This programming language is not a violation of the Church-Turing thesis – after some translation, the programming language can be executed on a normal computer.)

References

1. Bojańczyk, M., Braud, L., Klin, B., Lasota, S.: Towards nominal computation. In: POPL, pp. 401–412 (2012)
2. Bojańczyk, M., Klin, B., Lasota, S.: Automata with group actions. In: LICS, pp. 355–364 (2011)
3. Gabbay, M., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Asp. Comput. 13(3-5), 341–363 (2002)
4. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. 134(2), 329–363 (1994)

Dynamics of Rauzy Graphs for Low-Complexity Words

Julien Cassaigne

Institut de mathématiques de Luminy
case 907, 163, avenue de Luminy, 13288 Marseille Cedex 9
cassaigne@iml.univ-mrs.fr

Let $u \in A^{\mathbb{N}}$ be an infinite word. The *factor complexity* of u is the function $p\colon \mathbb{N} \to \mathbb{N}$ defined by: $p(n)$ is the number of words of length n occurring in u (*factors* of u). Morse and Hedlund [5] proved that $p(n) \geq n+1$ for non-eventually-periodic words. Words for which $p(n) = n+1$ are called Sturmian words. Words for which $p(n) = n+c$ for some constant c can be deduced from them [3]. We are interested in words "just above" this, roughly $n+1 \leq p(n) \leq 2n$. Let $\alpha_u = \liminf \frac{p(n)}{n}$ and $\beta_u = \limsup \frac{p(n)}{n}$, and $\Omega = \{(\alpha_u, \beta_u)\colon u \in A^{\mathbb{N}}\} \subseteq (\mathbb{R}^+ \cup \{+\infty\})^2$. Then the general problem, essentially open, is: what is the structure of Ω?

Heinis proved [4] that $\beta - \alpha \geq \frac{(2-\alpha)(\alpha-1)}{\alpha}$. In particular, $1 < \alpha = \beta < 2$ is impossible.[1] Aberkane [1] contructed a sequence of points of Ω converging to $(1,1)$; on the other hand, Turki [6] proved that $(\frac{3}{2}, \frac{5}{3})$ is an isolated point in Ω.

The main tool to study these words is the sequence of *Rauzy graphs*: Γ_n is the directed graph with vertices $L_n(u)$ (the factors of length n of u) and edges $L_{n+1}(u)$, with an edge from x to y labelled with z if and only if $z \in xA \cap Ay$. For Sturmian words, only two shapes of graphs are possible. For recurrent words with $p(n) \leq \frac{4}{3}n+1$, two new shapes appear. The language of such a word is then defined by a path in the *graph of shapes*, that controls how its Rauzy graphs evolve, and from which (α, β) and other properties may be deduced. The path also provides an *s-adic representation* (infinite composition of substitutions), which can be viewed as a generalized continued fraction expansion.

References

1. Aberkane, A.: Words whose complexity satisfies $\lim \frac{p(n)}{n} = 1$. Theoret. Comput. Sci. 307, 31–46 (2003)
2. Cassaigne, J., Nicolas, F.: Factor complexity. In: Berthé, V., Rigo, M. (eds.) Combinatorics, Automata and Number Theory, pp. 163–247. Cambridge University Press (2010)
3. Coven, E.M.: Sequences with minimal block growth II. Math. Systems Theory 8, 376–382 (1975)

[1] This is generalized in [2]: if $\lim \frac{p(n)}{n}$ exists, then it must be an integer.

4. Heinis, A.: The $P(n)/n$ function for bi-infinite words. Theoret. Comput. Sci. 273, 35–46 (2002)
5. Morse, M., Hedlund, G.A.: Symbolic Dynamics II. Sturmian trajectories. American J. Math. 62, 1–42 (1940)
6. Turki, R.: An isolated point in the Heinis spectrum (submitted manuscript, 2012)

Faster Algorithms for Sparse Fourier Transform

Piotr Indyk

MIT Computer Science and Artificial Intelligence Lab
32 Vassar Street
Cambridge, Massachusetts 02139

The Fast Fourier Transform (FFT) is one of the most fundamental numerical algorithms. It computes the Discrete Fourier Transform (DFT) of an n-dimensional signal in $\mathcal{O}(nlogn)$ time. The algorithm plays a key role in many areas.

In many applications (e.g., audio, image or video compression), most of the Fourier coefficients of a signal are "small" or equal to zero, i.e., the output of the transform is (approximately) sparse. In this case, there are algorithms that enable computing the non-zero coefficients faster than the FFT. However, in practice, the exponents in the runtime of these algorithms and their complex structure have limited their applicability to only very sparse signals.

In this talk, I will describe a new set of algorithms for sparse Fourier Transform. Their key feature is simplicity, which leads to efficient running time with low overhead, both in theory and in practice. In particular, one of the algorithms achieves a runtime of $\mathcal{O}(klogn)$, where k is the number of non-zero Fourier coefficients of the signal. This improves over the runtime of the FFT for any $k = \wr(n)$.

Joint work with Haitham Hassanieh, Dina Katabi and Eric Price.

Algorithms, Dichotomy and Statistics for Geometric and Sparse Graphs

Jaroslav Nešetřil

Department of Applied Mathematics
Faculty of Mathematics and Physics
Charles University, Prague

Sparse graphs present a problem: on the one side we often have better algorithms on the other side their structure and the lack of proper models makes the difficult to study. In this talk we present a new dichotomy "somewhere dense vs nowhere dense" classes a show the robustness and algorithmic relevance of this dichotomy. Particularly, we determine the asymptotic logarithmic density of subgraphs of large geometric graphs (and, more generally, of certain classes of sparse graphs). This leads to a unified approach to graph limits for both sparse and dense classes.

Joint work with Patrice Ossona de Mendez, Paris.

Table of Contents

Opening Talk

Full Papers

Can the Theory of Algorithms Ratify the "Invisible Hand of the Market"?

Vijay V. Vazirani

Georgia Institute of Technology
vazirani@cc.gatech.edu

Abstract.

It is not from the benevolence of the butcher, the brewer, or the baker, that we expect our dinner, but from their regard for their own interest. Each participant in a competitive economy is led by an invisible hand to promote an end which was no part of his intention.

Adam Smith, 1776.

With his treatise, The Wealth of Nations, 1776, Adam Smith initiated the field of economics, and his famous quote provided this field with its central guiding principle. The pioneering work of Walras (1874) gave a mathematical formulation for this statement, using his notion of market equilibrium, and opened up the possibility of a formal ratification.

Mathematical ratification came with the celebrated Arrow-Debreu Theorem (1954), which established existence of equilibrium in a very general model of the economy; however, an efficient mechanism for finding an equilibrium has remained elusive.

The latter question can clearly benefit from the powerful tools of modern complexity theory and algorithms. In this talk, we will provide an in-depth overview of the fascinating theory that has emerged around this question over the last decade.

1 Introduction

Following was the central question within mathematical economics for almost a century: Does a complex economy, with numerous goods and a large number of agents with diverse desires and buying powers, admit equilibrium prices? The mathematical study of this question was initiated by Walras in 1874 [28] when he gave a precise definition of such an economy and it culminated in the celebrated Arrow-Debreu Theorem [1] which provided an affirmative answer under some assumptions on the utility functions (they must satisfy non-satiation and be continuous and quasi-concave) and initial endowments of the agents (each agent must have a positive amount of each commodity); these are called *standard sufficient conditions* (though over the years, milder sufficient conditions were also obtained, see e.g. [20]). Finally, the Fundamental Theorems of Welfare Economics [2,8] made explicit the far reaching significance of a competitive equilibrium and

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 1–5, 2012.

provided a mathematical ratification of Adam Smith's notion of a free market economy.

Another fundamental question, that had been the subject of intense study within mathematical economics, was that of finding a mechanism or an algorithm for arriving at a market equilibrium. The search for a mechanism was also started by Walras [28], when he proposed his famous tatonnement process, and it essentially came to a halt with the Sonnenschein, Mantel and Debreu result showing that arbitrary excess demand functions could come from preference relations of the classical type, see [9]. We note that the proof of the Arrow-Debreu Theorem was based on Kakutani's fixed point theorem and alternative proofs are based on Brouwer's theorem; they are all therefore highly non-constructive.

Scarf [24] initiated the development of algorithms for computing market equilibria, introducing a family of procedures that compute approximate price equilibria by pivoting in a simplicial subdivision of the price simplex. A number of other methods, including Newton-based, homotopy methods, etc., have been developed in the following decades. These algorithms perform well in practice for several markets, but their running time is not polynomially bounded. The study of efficient computability of equilibria, from the perspective of modern theory of computation, was initiated by Megiddo and Papadimitriou [22]; see also Megiddo [21].

Over the last decade, the question of computability of market equilibria was taken up in full earnestness within theoretical computer science; in part this activity was also motivated by possible applications to markets on the Internet. This study concentrated on the two fundamental market models of Fisher [3] and Arrow-Debreu [1] (the latter is also known as the Walrasian model or the exchange model, and is more general than the Fisher model) under increasingly general and realistic utility functions.

Not surprisingly, the first results were for linear utility functions and the discovery of novel algorithmic ideas for handling this case for both market models led to much optimism [12,17]. Complexity results were also obtained for some specific non-linear utility functions that are well-studied in economics, e.g., Cobb-Douglas, CES, and Leontief [7,29,18]. In addition, this theory also defined new models that may be relevant to new markets and new issues, e.g., [25] defined the notion of spending constraint utilities and pointed out applications to the Adwords market, and [15] defined a perfect price discrimination market model which may be used in online display advertising marketplaces.

Within economics, concave utilities occupy a special place, since they capture the natural condition of decreasing marginal utilities. Hence, resolving their complexity has taken center stage over the last few years. Since we are dealing with a discrete computational model, it is natural to consider piecewise-linear, concave utilities. These can be further divided into two cases, non-separable and additively separable over goods; clearly, the latter is a subcase of the former.

The non-separable case contains Leontief utilities for which the following hardness results were obtained: for the Arrow-Debreu model, checking existence of an equilibrium is NP-hard, and for instances satisfying the standard Arrow-Debreu sufficient conditions, the computation of approximate equilibria is PPAD-hard [6,16,10]. However, if the number of goods is a constant, then a polynomial time algorithm exists for both market models [11].

This leaves the case of additively separable, piecewise-linear, concave (SPLC) utility functions. Recently, [4] made a breakthrough on this question by showing PPAD-hardness of computing equilibria, even approximate equilibria, for Arrow-Debreu markets with such utilities. Subsequently, PPAD-hardness for Fisher markets was proven independently by [5] and [27]. [27] also proved that for for case of SPLC utilities, the problem is in PPAD for both market models, hence exactly pinning down the complexity of this case. They also left the following open problem:

> *The definition of the class PPAD was designed to capture problems that allow for path following algorithms, in the style of the algorithms of Lemke-Howson [19] and Scarf [23]. Our result, showing membership in PPAD for both market models under separable, piecewise-linear, concave utility functions, establishes the existence of such path following algorithms for finding equilibria for these market models; however, it does so indirectly, by appealing to the characterization of PPAD given in [13]. It will be interesting to obtain natural, direct algorithms for this task (hence leading to a more direct proof of membership in PPAD), which may be useful for computing equilibria in practice.*

Using the powerful machinery of the linear complementarity problem and Lemke's algorithm, [14] gave such a path following algorithm for SPLC utilities for both Arrow-Debreu and Fisher markets. Even though their algorithm is exponential time in the worst case, it appears to be extremely fast in practice – experimental results on randomly generated instances suggest that the number of iterations needed is linear in the total number of segments (i.e., pieces) in all the utility functions specified in the input.

In summary, this decade-long endeavor has not only successfully revealed the broad outlines of the terrain of computability of market equilibria but has also contributed in a very substantial way to the fundamental theory of algorithms – not only new algorithmic ideas but also basic notions that transcend the originally intended applications. A specific example of the latter is the notion of a *rational convex program* [26]. At this point, the main question, of a general nature, remaining is to study the computability of equilibria of markets with production. Besides this, more work is needed on specific utility functions, especially those that find frequent applications and those involving non-separable utilities.

References

1. Arrow, K., Debreu, G.: Existence of an equilibrium for a competitive economy. Econometrica 22, 265–290 (1954)
2. Arrow, K.J.: An extension of the basic theorems of classical welfare economics. In: Proceedings of the Second Berkeley Symposium. University of California Press, Berkeley (1951)
3. Brainard, W.C., Scarf, H.E.: How to compute equilibrium prices in 1891. Cowles Foundation Discussion Paper (1270) (2000)
4. Chen, X., Dai, D., Du, Y., Teng, S.-H.: Settling the complexity of Arrow-Debreu equilibria in markets with additively separable utilities. In: FOCS (2009)
5. Chen, X., Teng, S.-H.: Spending Is Not Easier Than Trading: On the Computational Equivalence of Fisher and Arrow-Debreu Equilibria. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 647–656. Springer, Heidelberg (2009)
6. Codenotti, B., Saberi, A., Varadarajan, K., Ye, Y.: Leontief economies encode two-player zero-sum games. In: SODA (2006)
7. Codenotti, B., Varadarajan, K.R.: Efficient Computation of Equilibrium Prices for Markets with Leontief Utilities. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 371–382. Springer, Heidelberg (2004)
8. Debreu, G.: Theory of Value. Cowles Foundation Monograph (1959)
9. Debreu, G.: Excess demand functions. Journal of Mathematical Economics 1, 15–22 (1974)
10. Deng, X., Du, Y.: The computation of approximate competitive equilibrium is ppad-hard. Inform. Proc. Letters 108, 369–373
11. Devanur, N., Kannan, R.: Market equilibria in polynomial time for fixed number of goods or agents. In: FOCS, pp. 45–53 (2008)
12. Devanur, N., Papadimitriou, C.H., Saberi, A., Vazirani, V.V.: Market equilibrium via a primal-dual-type algorithm. JACM 55(5) (2008)
13. Etessami, K., Yannakakis, M.: On the complexity of Nash equilibria and other fixed points 39(6), 2531–2597 (2010)
14. Garg, J., Mehta, R., Sohoni, M., Vazirani, V.V.: A complementary pivot algorithm for market equilibrium under separable, piecewise-linear concave utilities. In: ACM Symposium on the Theory of Computing (2012)
15. Goel, G., Vazirani, V.V.: A perfect price discrimination market model with production and a rational convex program for it. Mathematics of Operations Research 36 (2011)
16. Huang, L.-S., Teng, S.-H.: On the Approximation and Smoothed Complexity of Leontief Market Equilibria. In: Preparata, F.P., Fang, Q. (eds.) FAW 2007. LNCS, vol. 4613, pp. 96–107. Springer, Heidelberg (2007)
17. Jain, K.: A polynomial time algorithm for computing the Arrow-Debreu market equilibrium for linear utilities. SIAM Journal on Computing 37(1), 306–318 (2007)
18. Jain, K., Vazirani, V.V., Ye, Y.: Market equilibrium for homothetic, quasi-concave utilities and economies of scale in production. In: SODA (2005)
19. Lemke, C.E., Howson Jr., J.T.: Equilibrium points of bimatrix games. SIAM Journal on Applied Mathematics 12(2), 413–423 (1964)
20. Maxfield, R.R.: General equilibrium and the theory of directed graphs. J. Math. Econ. 27(1), 23–51 (1997)
21. Megiddo, N.: A note on the complexity of P-matrix LCP and computing an equilibrium, iBM Research Report 6439 (1988), `http://theory.stanford.edu/~megiddo/pdf/plcp.pdf`

22. Megiddo, N., Papadimitriou, C.H.: On total functions, existence theorems, and computational complexity. Theoretical Computer Science 81, 317–324 (1991)
23. Scarf, H.: The approximation of fixed points of a continuous mapping. SIAM Journal on Applied Mathematics (15), 1328–1343 (1967)
24. Scarf, H.: The Computation of Economic Equilibria. Yale University Press (1973)
25. Vazirani, V.V.: Spending constraint utilities, with applications to the Adwords market. Mathematics of Operations Research 35(2) (2010)
26. Vazirani, V.V.: The notion of a rational convex program, and an algorithm for the Arrow-Debreu Nash bargaining game. Journal of the ACM 59(2) (2012)
27. Vazirani, V.V., Yannakakis, M.: Market equilibrium under separable, piecewise-linear, concave utilities. Journal of the ACM 58(3), 10:1–10:25 (2011)
28. Walras, L.: Éléments d'économie politique pure ou théorie de la richesse sociale (Elements of Pure Economics, or the theory of social wealth), Lausanne, Paris (1874); (4th ed., 1899; rev ed., 1926; Engl. transl., 1954)
29. Ye, Y.: Exchange market equilibria with leontief's utility: Freedom of pricing leads to rationality. Theoretical Computer Science 378, 134–142 (2007)

Resilient Quicksort and Selection

Maxim Babenko[1,2,⋆] and Ivan Pouzyrevsky[1,2,⋆⋆]

[1] Moscow State University, Russia
[2] Yandex, Russia

Abstract. We consider the problem of sorting a sequence of n keys in a RAM-like environment where memory faults are possible. An algorithm is said to be δ-*resilient* if it can tolerate up to δ memory faults during its execution. A resilient sorting algorithm must produce a sequence where every pair of uncorrupted keys is ordered correctly. Finocchi, Grandoni, and Italiano devised a δ-resilient deterministic mergesort algorithm that runs in $O(n \log n + \delta^2)$ time. We present a δ-resilient randomized algorithm (based on quicksort) that runs in $O(n \log n + \delta\sqrt{n \log n})$ expected time and its deterministic variation that runs in $O(n \log n + \delta\sqrt{n} \log n)$ worst-case time. This improves the previous known result for $\delta > \sqrt{n} \log n$.

Our deterministic sorting relies on the notion of an approximate k-th order statistic. For this auxiliary problem, we devise a deterministic algorithm that runs in $O(n + \delta\sqrt{n})$ time and produces a key (either corrupted or not) whose order rank differs from k by at most $O(\delta)$.

1 Introduction

1.1 Preliminaries

Recent trends in algorithm engineering indicate a rising demand for reliable computations. Applications that make use of large memory capacities at low cost face problems of memory faults [18,15]. Indeed, unpredictable failures known as *soft memory errors* tend to happen more often with the increase in memory size and speed [13,16]. Contemporary memory devices such as SRAM and DRAM units are unreliable due to a number of factors such as power failures, radiation, cosmic rays etc. The content of a cell in an unreliable memory can be silently altered and for standard memory circuits there is no direct way for detecting these types of corruptions.

Corrupted content in memory cells can affect many standard algorithms. For instance, during a typical binary search in a sorted array a single corruption encountered in an early stage of the search can cause the search path to end $\Omega(N)$ locations away from its correct position. Data replication can help to combat corruptions but is not always feasible since time and space overheads incurred by storing and fetching replicated values can be significant. Memory corruptions

⋆ maxim.babenko@gmail.com
⋆⋆ ivan.pouzyrevsky@gmail.com

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 6–17, 2012.

are of particular concern for applications dealing with massive amounts of data since such applications typically run for very long periods of time and are thus more likely to encounter memory cells containing corrupted data.

To design an algorithm that is provably resilient to memory corruptions, Finocchi and Italiano [12] introduced the *faulty memory random access machine*, which is based on a traditional RAM model. In this faulty model, memory corruptions can occur at *any time* and at *any place* in memory during the execution of an algorithm, and corrupted memory cells cannot be distinguished from uncorrupted cells. It is assumed that there is an adaptive adversary that chooses how, where, and when corruptions occur. This model has a parameter δ that is an upper bound on the number of corruptions the adversary can perform during a single run of the algorithm. Motivated by the fact that processor registers are considered incorruptible, $O(1)$ reliable memory locations are provided.

This model also extends to randomized computations, where, as defined in [11], the adversary does not see the random bits used by an algorithm. An algorithm is said to be *resilient* if it works *correctly* despite memory faults. The notion of correctness is stated explicitly for each possible kind of problem. For instance, a correct resilient sorting algorithm must output all uncorrupted elements in sorted order (while corrupted elements can appear at arbitrary positions in the output).

Throughout the paper we use the notion of a *trivially resilient storage*, which keeps its values in unreliable memory but still guarantees safety. This is achieved by replicating each stored value in $2\delta + 1$ consecutive cells. Since at most δ of copies can be corrupted, the majority of these $2\delta + 1$ elements remain uncorrupted. The correct value can be retrieved in $O(\delta)$ time and $O(1)$ space with the majority algorithm given [1] (which scans the copies keeping a single majority candidate and a counter in a reliable memory).

The above notion of trivial resiliency is extended to algorithms. Each algorithm that operates in a usual, non-faulty RAM model can be run in a faulty environment and produce correct results. To this aim all its memory (containing the input, the output, and the intermediate data) is kept in a trivially resilient storage. This trick, however, comes at a cost — it multiplies the complexity by δ. We shall be interested in alternative approaches that do not incur the δ-factor overhead.

1.2 Previous Work

Several important results were achieved in the faulty RAM model [9]. Concerning resilient dynamic data structures, search trees that support searches, insertions and deletions in $O(\log n + \delta^2)$ amortized time were introduced in [10]. A resilient priority queue was proposed in [14] supporting both insert and delete-min operations in $O(\log n + \delta)$ amortized time. Furthermore, in [2] a resilient dynamic dictionary implementing search, insert and delete operations in $O(\log n + \delta)$ expected amortized time was developed. Recently, Brodal et al. in [4] addressed the counting problem in faulty memory proposing a number of algorithms with various tradeoffs and guarantees.

For the sorting problem, the following definition is crucial (cf.[12,11]).

Definition 1. *A sequence is* faithfully ordered *if all its uncorrupted keys are sorted.*

Given a sequence S, a correct δ-resilient sorting algorithm must tolerate up to δ faults, i.e. produce a faithfully ordered sequence obtained by permuting the elements of S. (In presence of memory corruptions, getting a faithfully ordered sequence is the best we can hope for.) In [11] Finocchi and Italiano devised a deterministic δ-resilient algorithm that sorts n keys in $O(n \log n + \delta^2)$ time. They also proved in [12] that under certain additional assumptions a resilient comparison-based sorting algorithm must perform at least $\Omega(n \log n + \delta^{2-\varepsilon})$ comparisons to sort a sequence of length n when up to $\delta \leq n^{2/(3-2\varepsilon)}$ (for $\varepsilon \in [0, 1/2]$) corruptions may happen.

In [17,7,8] a number of empirical studies were conducted showing that resilient algorithms are of practical interest. The problem of combining external memory and resilient algorithms is considered in [3].

1.3 Our Contribution

Firstly, we address sorting problem. In contrast to a merge-based approach developed by Finocchi, Grandoni, and Italiano, we investigate how the divide-and-conquer strategy fits in the faulty memory model. In Section 2 we propose an $O(n \log n + \delta\sqrt{n \log n})$-time randomized δ-resilient algorithm that sorts n keys and its deterministic variation that runs in $O(n \log n + \delta\sqrt{n} \log n)$ worst-case time. For $\delta > \sqrt{n} \log n$, the method we propose is asymptotically faster than one in [11].

It may seem suspicious that our running time beats the lower bound from [12] but one should not worry since our algorithm employs explicit data replication and the analysis in [12] does not apply to this case. More details are given in Appendix. Based on this we stress that there is a possible gap between the actual complexity of the sorting problem in resilient setting and the bound proved in [12]. This also indicates that a careful usage of data replication may be a key to improving performance of resilient algorithms. In our case we were able to improve performance for large δ.

To devise deterministic variation of quicksort we had to deal with the selection problem. Hence in Section 3 we consider the problem of approximating the k-th order statistic in a faulty environment. That is, our selection algorithm runs in $O(n + \delta\sqrt{n})$ deterministic time, tolerates up to δ faults, and produces a key whose rank is off the desired k by at most $O(\delta)$. To the best of our knowledge, this is the first result of such kind.

2 Randomized Sorting

In this section we develop a randomized resilient sorting algorithm (denoted by Randomized-Resilient-Sort) that runs in $O(n \log n + \delta\sqrt{n \log n})$ expected time.

In contrast to Finocchi et. al. we develop a quicksort-like sorting algorithm. The main obstacle for an algorithm of such kind is the lack of a resilient stack: the adversary can mislead the recursive invocation and hence leave a part of the sequence unsorted or processed more than once. In order to overcome this problem, one could use a trivially resilient stack (i.e., as earlier, replicate each pushed value $2\delta + 1$ times and use majority selection during pops). Quicksort performs $\Theta(n)$ stack operations during its execution, which immediately leads to $\Theta(\delta n)$ overhead. We propose simple modifications to quicksort that balance resiliency and running time.

For our purposes the following notion will be of use:

Definition 2. *A routine that gets a sequence S of length n and computes some permuted sequence S' is said to be a* robust *sorting algorithm if the following properties are met: (i) if no memory faults occur during the execution, then the routine runs in $O(n \log n)$ time and returns a sorted sequence (ii) otherwise the routine still terminates in $O(n \log n)$ time (without any guarantees about the output).*

Note that robustness does not imply resilience. In fact, the output of a robust routine may be arbitrarily off from the desired (due to memory faults).

The standard quicksort sorting algorithm is not robust since it employs recursion. Recursive algorithm may behave unpredictably when run in a faulty memory environment since a single stack corruption may cause the algorithm to crash or, even worse, to enter an infinite loop. To overcome this issue, we maintain a watchdog timer (implemented as a tick counter stored in a reliable memory). If more than $cn \log n$ ticks elapse (where c is a sufficiently large constant chosen based on the worst-case analysis), the algorithm terminates prematurely. In this case at least one memory fault had occurred, so case (ii) from the definition applies. We denote the resulting robust sorting routine by Robust-Sort. An alternative approach to devise such a routine would be to use a non-recursive mergesort.

Randomized-Resilient-Sort is recursive; it takes a sequence S of length n and an additional parameter t. We choose $t := \lceil \sqrt{n/\log n} \rceil$ (where n is the length of the input at the top-most level of recursion). This t is treated as a fixed parameter for the rest of the algorithm.

If $n \leq t$, then apply Robust-Sort to S and validate its result by iterating over S and comparing each element with the previous one (keeping the latter in a reliable memory). If the sequence looks correctly ordered, then terminate Randomized-Resilient-Sort. Otherwise restart by applying Robust-Sort again.

Now suppose $n > t$. Like in a usual randomized quicksort, choose a pivot p in S (independently and uniformly) and perform Partition of S around p. Let the resulting subsequences be L (containing the values less than p) and R (containing the values greater than p). If either $|L| < n/4$ or $|R| < n/4$, then restart by picking another pivot and applying Partition again.

Algorithm 1. RANDOMIZED-RESILIENT-SORT(S, t)

```
1: Let n := |S|
2: if n ≤ t then
3:    repeat
4:       S' := ROBUST-SORT(S)
5:    until S' is ordered correctly
6:    return S'
7: else
8:    repeat
9:       Choose a random pivot p in S
10:      (L, R) := PARTITION(S, p)
11:   until |L| ≥ n/4 and |R| ≥ n/4
12:   L' := RANDOMIZED-RESILIENT-SORT(L, t)
13:   R' := RANDOMIZED-RESILIENT-SORT(R, t)
14:   return L' ∘ (p) ∘ R'
15: end if
```

Otherwise recurse to L and R to obtain sorted subsequences L' and R'. Use trivially resilient storage for storing stack frames during recursive invocations. Finally output the concatenation of L', p, and R'.

Theorem 1. RANDOMIZED-RESILIENT-SORT *produces a faithfully ordered sequence and runs in* $O(n \log n + \delta\sqrt{n \log n})$ *expected time.*

Proof. The fact that the resulting sequence is faithfully ordered is obvious, so we shall focus on estimating the time complexity.

Let n be the length of initial sequence S, n' be the length of current sequence in recursive invocation and also let n'' denote the length of the sequence in the parent call to RANDOMIZED-RESILIENT-SORT (if any). Consider case when $n' \le t$, which is handled in Steps 3–6. Note that $n'' > t$ and Step 11 ensures that $n' \ge n''/4 > t/4$. Hence during the whole computation Step 4 is applied to at most $4n/t$ distinct and non-overlapping ranges. Not counting restarts due to failed validation, this takes $O(t \log t \cdot 4n/t) = O(n \log n)$ time (since $t < n$). There are at most δ restarts of Step 4, each taking $O(t \log t) = O(\sqrt{n \log n})$ additional time.

Next consider the time spent to maintain the resilient stack. Due to the check in Step 11 the recursion tree is nearly-balanced. Each leaf deals with a range of length from $t/4$ to t, so (as indicated above) there are at most $4n/t$ leaf calls. The total number of nodes in the recursion tree is also $O(n/t)$, which is $O(\sqrt{n \log n})$. Therefore RANDOMIZED-RESILIENT-SORT performs $O(\sqrt{n \log n})$ push and pop operations in total, each taking $O(\delta)$ time.

It remains to bound the time spent in Steps 8–11. Each single PARTITION takes $O(n')$ time but there may be multiple restarts due to unbalanced sizes of L and R. Without memory faults, the uniform choice of the pivot ensures that the correct sizes are observed with probability about $1/2$ (see, e.g. [5]). Hence the expected number of repetitions is constant. When faults are possible, the

analysis becomes more intricate since the adversary may corrupt keys and thus force the algorithm to produce unbalanced partitions.

Without corruptions, it takes $O(1)$ attempts to find a pivot p with rank in $[3n'/8, 5n'/8]$. Now if the adversary corrupts less than $n'/8$ keys during this iteration, p's rank remains in $[n'/4, 3n'/4]$, so the check in Step 11 succeeds (here we use the fact that our random bits are unknown to the adversary). Assume the contrary, i.e. at least $\alpha = n'/8$ keys are corrupted (perhaps leading to an unbalanced partition). Such a "large corruption" costs $O(n') = O(\alpha)$ (sic!) time (before another good pivot whose rank is in $[3n'/8, 5n'/8]$ is picked). Summing over all large corruption cases and using the fact that the sum of αs is bounded by δ, one concludes that the additional overhead incurred by the adversary is $O(\delta)$, which is negligible.

The desired bound of $O(n \log n + \delta\sqrt{n \log n})$ follows by summing up the above estimates. □

3 Resilient Selection

In order to devise a deterministic variation of the quicksort we have to learn to select a good pivot (like a median). From our perspective, following notions will be of use:

Definition 3. *Let S be a sequence of n* distinct *items. Let $\mathrm{rk}_S(x) = \mathrm{rk}(x)$ denote the* rank *of x, i.e. the number of items in S that are less than or equal to x. The k-th order statistic of S is an element $x \in S$ such that $\mathrm{rk}_S(x) = k$. For $k = \lceil n/2 \rceil$, the k-th order statistic is referred to as the* median.

In a usual RAM model finding the k-th order statistic in a sequence S of length n is a widely studied problem. For instance, in [5] one can find a randomized $O(n)$ expected time algorithm and also a deterministic $O(n)$ worst-case time "median of medians" algorithm. Other methods are known that achieve better asymptotic constants for the number of comparisons, see e.g. [6].

In a faulty environment it is impossible to compute the order statistic exactly. Indeed, the adversary may corrupt keys just before the algorithm stops running thus destroying the correct answer. To overcome this issue, we relax the notion as follows:

Definition 4. *Let S be a sequence. Then x is a Δ-*approximate k-th order statistic *if $|\mathrm{rk}_S(x) - k| \leq \Delta$.*

Note that in the above definition x need not be an element of S. This observation is important since in the faulty memory model no algorithm can guarantee to output an item belonging to the initial S (as corrupted and uncorrupted values are indistinguishable). Extending the above argument one can see that for $\Delta < \delta$ computing a Δ-approximate k-th order statistic is impossible since the adversary can alter an element's rank for up to δ by corrupting elements less than or greater than the element.

Algorithm 2. PIVOT(S, m)

```
Let n := |S|
Split S into k := ⌈n/m⌉ chunks C_1, ..., C_k of length m or m + 1 each
for all chunks C_i do
    repeat
        Compute x_i := ROBUST-MEDIAN(C_i)
    until x_i passes validation as the median
    Put x_i into a trivially resilient storage
end for
return TRIVIALLY-RESILIENT-MEDIAN(x_1, ..., x_k)
```

There are algorithms for computing a Δ-approximate k-th order statistic. Trivially resilient approach would be to replicate each element in $2\delta + 1$ copies and run the standard linear time algorithm with resilient operations. This would result in $O(\delta n)$ time and $O(\delta n)$ extra space. In particular case of median selection, we will refer to this algorithm as TRIVIALLY-RESILIENT-MEDIAN.

Interestingly, if we allow $\Delta = O(\delta)$, then it is possible to solve the problem in $O(n + \delta\sqrt{n})$ worst-case deterministic time and $O(\delta\sqrt{n})$ space. Our method is based on the "median of medians" algorithm [5]. However, extra care is taken to make the algorithm resilient to memory faults.

As in Section 2 we define an analogous notion of robustness with the same guarantees. Specifically, robust median selection routine either computes a median in linear time when there are no memory faults or terminates prematurely in $O(n)$ time without any guarantees on the output in the presence of memory faults. To devise robust median selection routine we equip a standard algorithm with a watchdog timer (as in Section 2). We denote the resulting robust routine by ROBUST-MEDIAN.

We use ROBUST-MEDIAN (which runs in $O(n)$ time) and also TRIVIALLY-RESILIENT-MEDIAN (which runs in $O(\delta n)$ time) to construct a new procedure called PIVOT.

PIVOT gets a sequence S of length n and computes a value p as follows (a pseudocode shown in Algorithm 2). The input sequence is divided into k *chunks* of length either m or $m + 1$. This can be achieved by using a Bresenham-like partitioning of n. We treat m as a parameter and set $k := \lceil n/m \rceil$. For each chunk C_i, run ROBUST-MEDIAN, denote its output by x_i, and store the latter in a reliable memory. Due to memory faults x_i may differ from the true median. Run a *validation of* x_i, that is, traverse C_i and count the number of elements in C_i that are less than x_i. If $\mathrm{rk}_{C_i}(x_i) \neq \lfloor \frac{1}{2}|C_i| \rfloor$, then restart ROBUST-MEDIAN computation for this particular chunk C_i.

Finally we get a value x_i that "looks like" a true median of C_i. (Strictly speaking, x_i is the median of C_i', where C_i' consists of values of C_i that were observed by the algorithm during the validation pass.) At this point the algorithm stores x_i in a trivially resilient storage (i.e. with replication factor $2\delta+1$) and proceeds to the next chunk. When all the chunks are processed, we get a sequence of k

values $x_1, \ldots, x_k$ (each stored in a trivially resilient storage). The algorithm runs Trivially-Resilient-Median for these values and outputs the result.

Lemma 1. *Let α be the number of memory faults occurred during the execution. Assuming that $2k + m < n/5$,* Pivot *takes $O(n + \alpha m + \delta n/m)$ time and the resulting value p obeys*

$$\mathrm{rk}_S(p) \geq n/5 - \alpha \quad \textit{and} \quad \mathrm{rk}_S(p) \leq 4n/5 + \alpha \tag{1}$$

Proof. Pivot takes $O(km + \delta k) = O(n + \delta n/m)$ plus the time incurred by failed validation passes. There are at most α such passes, each taking $O(m)$ time. The total time bound follows.

To prove (1), consider the chunks $C_1, \ldots, C_k$ comprising S. For each chunk C_i, the algorithm computes (and stores in a trivially resilient storage) a value x_i, which is a candidate for the median of C_i that had passed validation. This way, x_i may be viewed as the median of some fixed sequence C'_i (consisting of values of C_i as they were observed by the algorithm during the validation pass). Such C'_i is in a sense "virtual": it may happen that at no moment of time the current C_i coincides with C'_i. Since $|C'_i|$ is either m or $m + 1$ there are at least $\lfloor m/2 \rfloor$ elements in C'_i that are less than or equal to x_i.

The returned value p is the median of $x_1, \ldots, x_k$ (the exact one since it is computed in a trivially resilient fashion). Therefore at least $\lfloor k/2 \rfloor$ values among $x_1, \ldots, x_k$ are less than or equal to p. Let S' denote the sequence obtained by concatenating $C'_1, \ldots, C'_k$. From the above estimates we can conclude that at least $\Delta = \lfloor m/2 \rfloor \cdot \lfloor k/2 \rfloor$ values in S' are less than or equal to p. Then $\Delta \geq (k-1)(m-1)/4 \geq (n - 2k - m + 1)/4 \geq n/5$ (since $2k + m < n/5$). Finally note that S and S' differ in at most α positions. The proof for lower bound follows. The proof for upper bound with can be obtained by reversing all inequalities with proper modifications. □

Note that in (1) ranks are considered w.r.t. the initial sequence S (before any corruptions took place). However one can easily see that (1) remains true if S denotes the values of the input sequence as they were observed at some (possibly different) moments of time. Indeed, in the above proof S and S' still differ in at most α positions (since each mismatch corresponds to a memory fault).

Corollary 1. *A pivot p satisfying (1) can be found in $O(n + \delta\sqrt{n})$ time.*

Proof. Choose $m := \lfloor \sqrt{n} \rfloor$. Observe that $k = O(\sqrt{n})$ and $\alpha \leq \delta$. We may assume that n is large enough so $2k + m < n/5$ and Lemma 1 applies. □

Now we are ready to present Resilient-Select (see Algorithm 3 for a pseudocode). It takes a sequence S of length n (which is assumed to be large enough), a parameter number k, and finds, in $O(n + \delta\sqrt{n})$ time, an $O(\delta)$-approximate k-th order statistic in S.

Three cases are to be considered, depending on the magnitude of n.

1. **Tiny case**: $n \leq 5\delta$. The algorithm outputs an arbitrary key, e.g. $S[1]$.

Algorithm 3. RESILIENT-SELECT(S, k)

```
1: Let n := |S|, m := ⌊√n⌋
2: if n ≤ 5δ then {tiny case}
3:    return S[1]
4: else if 5δ < n ≤ 20δ then {small case}
5:    return PIVOT(S, m)
6: else {large case}
7:    p := PIVOT(S, m)
8:    (L, R) := PARTITION(S, p)
9:    if |L| ≤ k then
10:      return RESILIENT-SELECT(L, k)
11:   else
12:      return RESILIENT-SELECT(R, k − |L|)
13:   end if
14: end if
```

2. **Small case**: $5\delta < n \le 20\delta$. Apply PIVOT to S (using $m := \lfloor\sqrt{n}\rfloor$) and output the resulting value.
3. **Large case**: $n > 20\delta$. First, invoke PIVOT (using, as above, $m := \lfloor\sqrt{n}\rfloor$) and denote the resulting value by p. Second, perform PARTITION of S around p into subsequences L and R; that is, enumerate the elements of S and put those not exceeding p into L and the others into R. (For simplicity's sake, we assume that all elements are distinct. For coinciding elements, a more careful choice between L and R is needed to avoid unbalanced partitions. These details are quite technical and we omit them due to the lack of space.) These two new sequences L and R are stored in the usual, faulty memory. Their lengths $|L|$ and $|R|$, however, are stored in reliable (safe) memory. Third, if $k \le |L|$, then recurse to L. Otherwise reset $k := k - |L|$ and recurse to R. (These steps are standard for divide-and-conquer selection algorithms.) The tail recursion in RESILIENT-SELECT can be easily eliminated, hence there is no need for a stack.

Theorem 2. RESILIENT-SELECT *runs in* $O(n+\delta\sqrt{n})$ *time and returns an* $O(\delta)$*-approximate* k*-th order statistic.*

Proof. Let S be the input sequence where an approximate k-th order statistic is to be found. Due to memory faults the sequence may be altered during execution; we shall denote its current state by S'.

For the tiny case, any returned value would satisfy as approximation. For the small and large case we use induction to show that the returned value p is indeed a 20δ-approximate k-order statistic. Specifically, we claim that two inequalities hold (where α is the number of faults occurred during the invocation):

$$\min S \le p \le \max S \quad \text{and} \quad |\mathrm{rk}_S - k| \le 20\alpha.$$

The small case is an inductive base. It follows from Lemma 1 and Corollary 1 that $\mathrm{rk}_S(p) \ge n/5 - \alpha \ge 1$ and $\mathrm{rk}_S(p) \le 4n/5 + \alpha < 20\delta \le n$. Hence p is

between $\min S$ and $\max S$. For any $x \in S$, $|\mathrm{rk}_S(p) - \mathrm{rk}_S(x)| \leq 4n/5 + \alpha \leq 20\alpha$. In particular $|\mathrm{rk}_S(p) - k| \leq 20\alpha$, as desired.

Now consider a large case. Note that PARTITION reads each element of S exactly once. Hence one may imagine a sequence S' consisting of values in S as they were observed by PARTITION. Also we may assume that L and R form a partition of S' (not S) around p. Indeed, all memory faults that occur in L or R and affect the values written by PARTITION may be effectively "postponed" until RESILIENT-SELECT calls itself recursively. In other words, these faults are regarded as occurring in the latter recursive invocation.

Let γ (respectively β) be the total number of faults occurred in RESILIENT-SELECT from start until the recursive invocation is made (respectively during the recursive invocation). Suppose RESILIENT-SELECT recurses to L. Then by the inductive assumption the returned value x satisfies $|\mathrm{rk}_L(x) - k| \leq 20\beta$. Observe the equality $\mathrm{rk}_L(x) = \mathrm{rk}_{S'}(x)$ (here we use the fact that $\min L \leq x \leq \max L$). Altogether this implies $|\mathrm{rk}_{S'}(x) - k| \leq 20\beta$.

Sequences S and S' differ by at most γ elements (the latter is the upper bound for the number of memory faults occurred during RESILIENT-SELECT not counting the recursion). Therefore $|\mathrm{rk}_S(x) - \mathrm{rk}_{S'}(x)| \leq \gamma$, so $|\mathrm{rk}_S(x) - k| \leq 20\beta + \gamma \leq 20(\beta + \gamma) = 20\delta$.

The case when RESILIENT-SELECT recurses to R is analogous.

Finally let us estimate the complexity of RESILIENT-SELECT. The i-th level of recursion, which is applied to a sequence of length n_i, takes $O(n_i + \delta\sqrt{n_i})$ time (see Corollary 1). Summing over all levels one gets $O\left(\sum_i n_i + \delta \sum_i \sqrt{n_i}\right)$. Each recursive call reduces the current length n_i to $n_{i+1} \leq 4n_i/5 + \delta \leq 4n_i/5 + n_i/20 = 17n_i/20$. Since the decrease is exponential, the estimate becomes $O(n + \delta\sqrt{n})$, as claimed. □

Corollary 2. *There exists a deterministic $O(n)$-time $O(\delta)$-approximate median selection algorithm that tolerates up to $\delta = O(\sqrt{n})$ memory faults.*

4 Deterministic Sorting

Now we are ready to present a deterministic variation of quicksort (denoted by DETERMINISTIC-RESILIENT-SORT) with $O(n \log n + \delta\sqrt{n} \log n)$ worst-case running time. We effectively combine ideas from Section 2 and Section 3.

Let n be the length of the input sequence (which is assumed to be sufficiently large). The outline is the same as in the randomized algorithm in Section 2 except for three aspects. First, at the top level we choose $t := \lceil \sqrt{n} \rceil$ (in contrast to $t := \lceil \sqrt{n/\log n} \rceil$ in the randomized approach). Second, instead of selecting a random pivot we use deterministic PIVOT procedure in conjunction with PARTITION to ensure that resulting partition is nearly-balanced (i.e. there are at least 1/10 of elements in each part). Third, we perform a fallback to ROBUST-SORT when the length of the current segment becomes smaller than $15t$ rather than t. The latter is done to satisfy the preconditions of Lemma 1, namely, to make sure that

Algorithm 4. DETERMINISTIC-RESILIENT-SORT(S, t)

```
1: Let n := |S|
2: if n ≤ 15t then
3:    repeat
4:       S' := ROBUST-SORT(S)
5:    until S' is ordered correctly
6:    return S'
7: else
8:    repeat
9:       Let p := PIVOT(S, t)
10:      (L, R) := PARTITION(S, p)
11:   until |L| ≥ n/10 and |R| ≥ n/10
12:   L' := DETERMINISTIC-RESILIENT-SORT(L, t)
13:   R' := DETERMINISTIC-RESILIENT-SORT(R, t)
14:   return L' ∘ (p) ∘ R'
15: end if
```

$2t + \lceil n'/t \rceil < n'/5$ for the current length n'. (Indeed, $2t + \lceil n'/t \rceil \le 3t < n'/5$ when $n' > 15t$.) For a pseudocode, see Algorithm 4.

Theorem 3. DETERMINISTIC-RESILIENT-SORT *produces a faithfully ordered sequence and runs in* $O(n \log n + \delta\sqrt{n}\, \log n)$ *deterministic time.*

Proof. Once again, the fact that the resulting sequence is faithfully ordered is obvious, so we focus on estimating the time complexity.

Consider the recursion tree of DETERMINISTIC-RESILIENT-SORT. The latter is nearly-balanced (by the same argument as in the proof of Theorem 1) and thus contains $O(n/t)$ leaves. Hence ROBUST-SORT, which is invoked at the bottom level, costs $O(n/t \cdot t \log t + \delta t \log t)$ time.

At each level of the recursion tree some calls to PIVOT and PARTITION are made. Since PARTITION takes linear time, it contributes $O(n \log n)$ plus the time spent in additional restarts incurred by failed validations. To estimate PIVOT's total time, let N be the total length of segments for which PIVOT is invoked. Clearly $N = O(n \log n)$ (since the tree is nearly-balanced). Then by Lemma 1 all PIVOTs take $O(N + \delta t + \delta N/t)$ time (not counting additional restarts).

Consider additional restarts occurring in Steps 8–11. As in in the proof of Theorem 1, without corruptions PIVOT returns p whose rank is in $[n'/5, 4n'/5]$ (where n' is the current length of the segment). Now if the adversary corrupts less than $n'/10$ keys during the iteration, p's rank remains in $[n'/10, 9n'/10]$, so the check in Step 11 succeeds. Assume the contrary, i.e. at least $\alpha = n'/10$ keys are corrupted (perhaps leading to an unbalanced partition). Such a "large corruption" increases N by 10α, which adds up to $O(\delta)$. Hence the overhead is $O(\delta + \delta^2/t)$, which is negligible.

Summing up these estimates and plugging in $t = \sqrt{n}$ yields the desired time bound. □

References

1. Boyer, R.S., Moore, J.S.: MJRTY - A Fast Majority Vote Algorithm (1982)
2. Brodal, G.S., Fagerberg, R., Finocchi, I., Grandoni, F., Italiano, G.F., Jørgensen, A.G., Moruz, G., Mølhave, T.: Optimal Resilient Dynamic Dictionaries. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 347–358. Springer, Heidelberg (2007)
3. Brodal, G.S., Jørgensen, A.G., Mølhave, T.: Fault Tolerant External Memory Algorithms. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 411–422. Springer, Heidelberg (2009)
4. Brodal, G.S., Jørgensen, A.G., Moruz, G., Mølhave, T.: Counting in the Presence of Memory Faults. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 842–851. Springer, Heidelberg (2009)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd revised edn. The MIT Press (2001)
6. Dor, D.: Selection Algorithms. Ph.D. thesis, Tel-Aviv University (1995)
7. Ferraro-Petrillo, U., Finocchi, I., Italiano, G.: Experimental Study of Resilient Algorithms and Data Structures. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 1–12. Springer, Heidelberg (2010)
8. Ferraro-Petrillo, U., Grandoni, F., Italiano, G.: Data Structures Resilient to Memory Faults: An Experimental Study of Dictionaries. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 398–410. Springer, Heidelberg (2010)
9. Finocchi, I., Grandoni, F., Italiano, G.: Designing reliable algorithms in unreliable memories. Computer Science Review 1(2), 77–87 (2007)
10. Finocchi, I., Grandoni, F., Italiano, G.F.: Resilient search trees. In: Proc. SODA 2007, pp. 547–553. Society for Industrial and Applied Mathematics (2007)
11. Finocchi, I., Grandoni, F., Italiano, G.F.: Optimal resilient sorting and searching in the presence of memory faults. Theor. Comput. Sci. 410, 4457–4470 (2009)
12. Finocchi, I., Italiano, G.F.: Sorting and searching in the presence of memory faults (without redundancy). In: Proc. STOC 2004, pp. 101–110. ACM (2004)
13. Hamdioui, S., Ars, Z.A., Van De Goor, A.J., Rodgers, M.: Dynamic Faults in Random-Access-Memories: Concept, Fault Models and Tests. J. Electron. Test. 19, 195–205 (2003)
14. Jørgensen, A.G., Moruz, G., Mølhave, T.: Priority Queues Resilient to Memory Faults. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 127–138. Springer, Heidelberg (2007)
15. Li, X., Huang, M.C., Shen, K., Chu, L.: A realistic evaluation of memory hardware errors and software system susceptibility. In: Proc. USENIX 2010. USENIX Association (2010)
16. May, T.C., Woods, M.H.: Alpha-particle-induced soft errors in dynamic memories. IEEE Transactions on Electron Devices 26(1), 2–9 (1979)
17. Ferraro-Petrillo, U., Finocchi, I., Italiano, G.F.: The Price of Resiliency: A Case Study on Sorting with Memory Faults. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 768–779. Springer, Heidelberg (2006)
18. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. In: DSN 2006: Proceedings of the International Conference on Dependable Systems and Networks (DSN 2006), pp. 249–258. IEEE Computer Society, Los Alamitos (2006)

General Quantitative Specification Theories with Modalities

Sebastian S. Bauer[1], Uli Fahrenberg[2], Axel Legay[2], and Claus Thrane[3]

[1] Ludwig-Maximilians-Universität München, Germany
[2] Irisa/INRIA Rennes, France
[3] Aalborg University, Denmark

Abstract. This paper proposes a new theory of quantitative specifications. It generalizes the notions of step-wise refinement and compositional design operations from the Boolean to an arbitrary quantitative setting. It is shown that this general approach permits to recast many existing problems which arise in system design.

1 Introduction

Specification theories permit reasoning about behaviors of systems at the abstract level, which is needed in various application such as abstraction-based model checking for programming languages, or compositional reasoning. Such specification theories generally come with (1) a satisfaction relation that allows to decide whether an implementation is a model of the specification, (2) a notion of refinement for determining the relationship between specifications and their sets of implementations, (3) a structural composition which, at the abstract level, mimics the behavioral composition of systems, (4) a quotient that allows to synthesize specifications from refinements, and (5) a logical composition that allows to compute intersections of sets of implementations.

Prominent among specification theories is the one of *modal transition systems* [13,14,15,18,21], which are labeled transition systems equipped with two types of transitions: *must* transitions that are mandatory for any implementation, and *may* transitions which are optional. In recent work [3,17], modal transition systems have been extended by adding richer information to the usual discrete label set of transition systems, permitting to reason about *quantitative* aspects of models and specifications. These quantitative labels can be used to model and analyze *e.g.* timing [6,16], resource usage [22], or energy consumption [4,9].

In particular, [17] extends modal transition systems with integer intervals and introduces corresponding extensions of the above operations which observe the added quantitative information, and [3] generalizes this theory to general *structured labels*. Both theories are, however, *fragile* in the sense that they rely on Boolean notions of satisfaction and refinement: as refinement either holds or does not, they are unable to *quantify* the impact of small variations. For quantifying differences, *distances* between systems are useful; this approach has been

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 18–30, 2012.

explored *e.g.* in [5,7,19,23,25]. A first quantitative specification theory which is not fragile is introduced in [2], for one specific type of weighted modal transition systems and one specific distance. While this is useful for some applications, it is too specific to cover the whole spectrum of quantitative specification theories.

What is needed is a quantitative specification theory that is independent of both the specific labels and the distance used to measure differences; this is what we introduce in this paper. Using the concept of distance iterator function from [10,12], we introduce a general notion of refinement distance between structured modal transition systems and a general quantitative specification theory. It turns out that there are some natural technical compatibility conditions relating the label composition operators with the distance which give rise to different properties of the specification theory; these are worked out in detail.

Our general quantitative theory can be instantiated with a variety of different distances and operators, all useful for different applications; hence it can serve as a unifying framework for these applications. Note that all proofs of this paper had to be omitted due to space constraints.

2 Structured Modal Transition Systems

Labeled transition systems have long been established as the de-facto formalism for specifying formal semantics for discrete behavior and communication of programming languages and reactive systems. However, in order to capture meta-data and expectations about these, such as *e.g.* execution times of hardware platforms, cost of certain operations, or energy consumption, we require a richer formalism.

We work with a poset Spec of *specification labels* with a partial order $\sqsubseteq_{\mathsf{Spec}}$ and denote by $\mathsf{Spec}^{\infty} = \mathsf{Spec}^{*} \cup \mathsf{Spec}^{\omega}$ the set of finite and infinite traces over Spec. In applications, Spec may be used to model data about the behavior of a system; for specifications this may be considered as legal parameters of operation, whereas for implementations it may be thought of as observed information. The partial order $\sqsubseteq_{\mathsf{Spec}}$ is meant to model *refinement* of data; if $k \sqsubseteq_{\mathsf{Spec}} \ell$, then k is more refined (leaves fewer choices) than ℓ. The set $\mathsf{Imp} = \{k \in \mathsf{Spec} \mid k' \sqsubseteq_{\mathsf{Spec}} k \implies k' = k\}$ is called the set of *implementation labels*; these are the data which cannot be refined further. We let $[\![k]\!] = \{k' \in \mathsf{Imp} \mid k' \sqsubseteq k\}$ and assume that Spec is well-formed in the sense that $[\![k]\!] \neq \emptyset$ for all $k \in \mathsf{Spec}$.

When $k \not\sqsubseteq_{\mathsf{Spec}} \ell$, we want to be able to quantify the impact of this difference in data on the systems in question, thus circumventing the fragility of the theory. To this end, we introduce a general notion of distance on sequences of data following the approach laid out in [12]. Let M be an arbitrary set and $\mathbb{L} = (\ _{\geq 0} \cup \{\infty\})^M$ the set of functions from M to the extended non-negative real line. Then $\mathbb{L}$ is a complete lattice with partial order $\sqsubseteq\ $ given by $\alpha \sqsubseteq\ \beta$ if and only if $\alpha(x) \leq \beta(x)$ for all $x \in M$, and with an addition $\oplus\ $ given by $(\alpha \oplus\ \beta)(x) = \alpha(x) + \beta(x)$. The bottom element of $\mathbb{L}$ is also the zero of $\oplus\ $ and given by $\bot\ (x) = 0$, and the top element is $\top\ (x) = \infty$. We also define a metric on $\mathbb{L}$ by $d\ (\alpha, \beta) = \sup_{x \in M} |\alpha(x) - \beta(x)|$.

Let $d : \mathsf{Imp} \times \mathsf{Imp} \to \mathbb{L}$ be a hemimetric on implementation labels; recall that this means that $d(m,m) = \bot$ for all $m \in \mathsf{Imp}$ and $d(m_1,m_2) \oplus d(m_2,m_3) \sqsupseteq d(m_1,m_3)$ (the *triangle inequality*) for all $m_1, m_2, m_3 \in \mathsf{Imp}$. We extend d to Spec by $d(k,\ell) = \sup_{m \in [\![k]\!]} \inf_{n \in [\![\ell]\!]} d(m,n)$. Hence also this distance is *asymmetric*; the intuition is that any label in $[\![k]\!]$ has to be matched as good as possible in $[\![\ell]\!]$. Note the similarity of this to the construction of *Hausdorff metric*; *cf.* [20,1] for background material on hemimetrics and the Hausdorff construction.

We will assume given an abstract *trace distance* $d_T : \mathsf{Spec}^\infty \times \mathsf{Spec}^\infty \to \mathbb{L}$ which has a recursive expression using a *distance iterator* function $F : \mathsf{Imp} \times \mathsf{Imp} \times \mathbb{L} \to \mathbb{L}$. This will allow us to recover many of the system distances found in the literature, while preserving key results. We will need F to be *continuous* in the first two coordinates and *monotone* in the third; hence $F(\cdot, n, \alpha)$ and $F(m, \cdot, \alpha)$ are continuous functions $\mathsf{Imp} \to \mathbb{L}$ for all $\alpha \in \mathbb{L}$,and $F(m,n,\cdot) : \mathbb{L} \to \mathbb{L}$ is monotone for all $m, n \in \mathsf{Imp}$.

We also assume that $F(m,n,\bot) = d(m,n)$ for all $m, n \in \mathsf{Imp}$, and we extend F to specification labels by defining $F(k,\ell,\alpha) = \sup_{m \in [\![k]\!]} \inf_{n \in [\![\ell]\!]} F(m,n,\alpha)$. Then also the extended $F : \mathsf{Spec} \times \mathsf{Spec} \times \mathbb{L} \to \mathbb{L}$ is continuous in the first two and monotone in the third coordinates. Additionally, we assume that sets of implementation labels are *closed* with respect to F in the sense that for all $k, \ell \in \mathsf{Spec}$ and $\alpha \in \mathbb{L}$ with $F(k,\ell,\alpha) \neq \top$, there are $m \in [\![k]\!]$, $n \in [\![\ell]\!]$ with $F(m,\ell,\alpha) = F(k,n,\alpha) = F(k,\ell,\alpha)$. Note that this implies that the sets $[\![k]\!]$ are closed under the hemimetric d on Spec. We also extend the triangle inequality for d to F by imposing that for all $k, \ell, m \in \mathsf{Spec}$ and $\alpha, \beta, \gamma \in \mathbb{L}$ with $\alpha \oplus \beta \sqsupseteq \gamma$,

$$F(k,\ell,\alpha) \oplus F(\ell,m,\beta) \sqsupseteq F(k,m,\gamma). \tag{1}$$

Let $\boldsymbol{\varepsilon} \in \mathsf{Spec}^\infty$ denote the empty sequence, and for any sequence $\sigma \in \mathsf{Spec}^\infty$, denote by σ_0 its first element and by σ^1 the tail of the sequence with the first element removed. We assume that d_T has a recursive characterization, using F, as follows:

$$d_T(\sigma,\tau) = \begin{cases} F(\sigma_0, \tau_0, d_T(\sigma^1, \tau^1)) & \text{if } \sigma, \tau \neq \boldsymbol{\varepsilon}, \\ \top & \text{if } \sigma = \boldsymbol{\varepsilon}, \tau \neq \boldsymbol{\varepsilon} \text{ or } \sigma \neq \boldsymbol{\varepsilon}, \tau = \boldsymbol{\varepsilon}, \\ \bot & \text{if } \sigma = \tau = \boldsymbol{\varepsilon}. \end{cases} \tag{2}$$

In applications (see below), the lattice $\mathbb{L}$ comes equipped with a homomorphism $g : \mathbb{L} \to \ _{\geq 0} \cup \{\infty\}$ for which $g(d_T(\sigma,\sigma)) = 0$ for all $\sigma \in \mathsf{Spec}^\infty$. The actual trace distance of interest is then the composition $g \circ d_T$. The triangle inequality for F implies the usual triangle inequality for $g \circ d_T$: $g(d_T(\sigma,\tau)) + g(d_T(\tau,\chi)) \leq g(d_T(\sigma,\chi))$ for all $\sigma, \tau, \chi \in \mathsf{Spec}^\infty$, hence $g \circ d_T$ is a hemimetric on Spec^∞. We need to work with distances which factor through $\mathbb{L}$, instead of plainly taking values in $_{\geq 0} \cup \{\infty\}$, because some distances which are useful in practice, as the one in Example 2 below, have no recursive characterization using $\mathbb{L} = \ _{\geq 0} \cup \{\infty\}$. Whether the theory works for more general intermediate lattices than $L = (\ _{\geq 0} \cup \{\infty\})^M$ is an open question; we have had no occasion to use more general lattices in practice.

Example 1. [2] defines an *accumulating* distance for integer-weighted modal transition systems. In this paper, $\mathsf{Spec} = \Sigma \times \mathbb{I}$, where Σ is a finite set of discrete labels and $\mathbb{I} = \{[l,r] \mid l \in \quad \cup \{-\infty\}, r \in \quad \cup \{\infty\}, l \leq r\}$ is the set of extended-integer intervals, and the partial order is defined by $(a,[l,r]) \sqsubseteq_{\mathsf{Spec}} (a',[l',r'])$ if and only if $a = a'$, $l' \leq l$, and $r' \geq r$. Hence refinement is given by restricting intervals, thus $\mathsf{Imp} = \Sigma \times \quad$. The implementation label distance is given by $d((a,x),(a',y)) = |x-y|$ if $a = a'$ and ∞ otherwise.

Now let $\mathbb{L} = \quad_{\geq 0} \cup \{\infty\}$ and $F(m,n,\alpha) = d(m,n) + \lambda\alpha$ for some fixed *discounting factor* $\lambda \in \quad$ with $0 < \lambda < 1$, then $d_T(\sigma,\tau) = \sum_j \lambda^j d(\sigma_j,\tau_j)$ for implementation traces of equal length. This distance hence accumulates individual distances on labels; it has been studied for weighted transition systems and games *e.g.* in [5,7,8,19,23,25,26]. [2] develops a complete specification theory around this specific distance; we will continue this example below to show how it fits in our present context.

Example 2. With the same instantiations of Imp and Spec as above, we can introduce a distance which, instead of accumulating individual label differences, measures the long-run difference between *accumulated labels*. This *maximum-lead* distance is especially useful for real-time systems and has been considered in [16,23].

Let $\mathbb{L} = (\quad_{\geq 0} \cup \{\infty\})^{\mathbb{R}}$, define $F : \mathsf{Imp} \times \mathsf{Imp} \times \mathbb{L} \to \mathbb{L}$ by $F((a,x),(a',y),\alpha) = \top$ if $a \neq a'$ and $F((a,x),(a,y),\alpha)(\delta) = \max(|\delta + x - y|, \alpha(\delta + x - y))$, and extend F to specifications by $F(k,\ell,\alpha) = \sup_{m \in [\![k]\!]} \inf_{n \in [\![\ell]\!]} F(m,n,\alpha)$. Define $g : \mathbb{L} \to \quad_{\geq 0} \cup \{\infty\}$ by $g(\alpha) = \alpha(0)$; the maximum-lead distance assuming the lead is zero. Using our definition of d_T from (2), it can then be shown that for implementation traces $\sigma = ((a_0,x_0),(a_1,x_1),\dots)$, $\tau = ((a_0,y_0),(a_1,y_1),\dots)$, $g(d_T(\sigma,\tau)) = \sup_m |\sum_{i=0}^m x_i - \sum_{i=0}^m y_i|$ is precisely the maximum-lead distance of [16,12].

A *structured modal transition system* (SMTS) is a tuple $(S, s_0, \dashrightarrow_S, \longrightarrow_S)$ consisting of a set S of states, an initial state $s_0 \in S$, and *must* and *may* transitions $\longrightarrow_S, \dashrightarrow_S \subseteq S \times \mathsf{Spec} \times S$ for which it holds that for all $s \xrightarrow{k}_S s'$ there is $s \overset{\ell}{\dashrightarrow}_S s'$ with $k \sqsubseteq_{\mathsf{Spec}} \ell$. This last condition is one of *consistency*: everything which is required, is also allowed. S is an *implementation* if $\longrightarrow_S = \dashrightarrow_S \subseteq S \times \mathsf{Imp} \times S$; hence in an implementation, all optional behavior has been resolved, and all data has been refined to implementation labels.

We will assume all SMTS to be *compactly branching* [24], that is, for any SMTS S and any $s \in S$, the sets $\{k \in \mathsf{Spec} \mid s \overset{k}{\dashrightarrow} s'\}$ and $\{k \in \mathsf{Spec} \mid s \xrightarrow{k} s'\}$ are to be compact under the label metric d. This is a common assumption in quantitative formalisms which generalizes the standard finitely-branching assumption; together with continuity of F it will allow us to resolve terms of the form $\sup_{s \overset{k}{\dashrightarrow} s'} \inf_{t \overset{\ell}{\dashrightarrow} t'} F(k,\ell,\alpha)$.

The SMTS $(S, s_0, \dashrightarrow_S, \longrightarrow_S)$ is *deterministic* if it holds for all $s \in S$, $s \overset{k_1}{\dashrightarrow}_S s_1$, $s \overset{k_2}{\dashrightarrow}_S s_2$ for which there is $k \in \mathsf{Spec}$ with $d(k,k_1) \neq \top$ and $d(k,k_2) \neq \top$ that $k_1 = k_2$ and $s_1 = s_2$.

Example 1 (contd). For the label distance of [2], and also the one of Example 2, the above condition that there exist $k \in \mathsf{Spec}$ with $d(k, k_1) \neq \top$ and $d(k, k_2) \neq \top$ is equivalent, with $k_1 = (a_1, I_1)$ and $k_2 = (a_2, I_2)$, to saying that $a_1 = a_2$, hence our notion of determinism agrees with the one of [2]. The general intuition for determinacy is that there cannot be two distinct transitions out of a state with labels which have a common quantitative refinement.

A *modal refinement* of SMTS S, T is a relation $R \subseteq S \times T$ such that for any $(s,t) \in R$,

- whenever $s \overset{k}{\dashrightarrow}_S s'$, then also $t \overset{\ell}{\dashrightarrow}_T t'$ for some $k \sqsubseteq_{\mathsf{Spec}} \ell$ and $(s',t') \in R$,
- whenever $t \overset{\ell}{\longrightarrow}_T t'$, then also $s \overset{k}{\longrightarrow}_S s'$ for some $k \sqsubseteq_{\mathsf{Spec}} \ell$ and $(s',t') \in R$.

Thus any behavior which is permitted in S is also permitted in T, and any behavior required in T is also required in S. We write $S \leq_m T$ if there is a modal refinement $R \subseteq S \times T$ with $(s_0, t_0) \in R$. The *implementation semantics* of a SMTS S is the set $[\![S]\!] = \{I \leq_m S \mid I \text{ is an implementation}\}$, and we write $S \leq_t T$ if $[\![S]\!] \subseteq [\![T]\!]$, saying that S thoroughly refines T.

3 Refinement Distances

We define two distances between SMTS, one at the syntactic and one at the semantic level. The *modal refinement distance* $d_m : S \times T \to \mathbb{L}$ between the states of SMTS S, T is defined to be the least fixed point to the equations

$$d_m(s,t) = \max \begin{cases} \sup\limits_{s \overset{k}{\dashrightarrow}_S s'} \inf\limits_{t \overset{\ell}{\dashrightarrow}_T t'} F(k, \ell, d_m(s', t')), \\ \sup\limits_{t \overset{\ell}{\longrightarrow}_T t'} \inf\limits_{s \overset{k}{\longrightarrow}_S s'} F(k, \ell, d_m(s', t')). \end{cases}$$

We let $d_m(S,T) = d_m(s_0, t_0)$, and we write $S \leq_m^\alpha T$ if $d_m(S,T) \sqsubseteq \alpha$. This definition is an extension of the one of *simulation distance* in [12], and the proof of existence of the least fixed point is similar to the one in [12]. Note also that d_m extends the refinement relation $\leq_m$ in the sense that $s \leq_m t$ implies $d_m(s,t) = 0$. If we define the *linear distance* from s to t by $d_T(s,t) = \max\{\sup_{\sigma \in \mathrm{Tr}(s)} \inf_{\tau \in \mathrm{Tr}(t)} d_T(\sigma, \tau), \sup_{\tau \in \mathrm{Tr}(t)} \inf_{\sigma \in \mathrm{Tr}(s)} d_T(\sigma, \tau)\}$, where $\mathrm{Tr}(s)$ denotes the set of (*may* or *must*) traces emanating from s, then $d_T(s,t) \sqsubseteq d_m(s,t)$ for all $s, t \in S$, *cf.* [12].

The *thorough refinement distance* from an SMTS S to an SMTS T is

$$d_t(S,T) = \sup_{I \in [\![S]\!]} \inf_{J \in [\![T]\!]} d_m(I,J),$$

and we write $S \leq_t^\alpha T$ if $d_t(S,T) \sqsubseteq \alpha$. Again, $S \leq_t T$ implies $d_t(S,T) = 0$. The next proposition follows directly from the triangle inequality (1).

Proposition 1. *For all SMTS S, T, U, $d_m(S,T) \oplus d_m(T,U) \sqsupseteq d_m(S,U)$ and $d_t(S,T) \oplus d_t(T,U) \sqsupseteq d_t(S,U)$.*

The next theorem shows that the modal refinement distance overapproximates the thorough one, and that it is exact for deterministic SMTS. This is similar to the situation for standard modal transition systems [18]; note [18] that deterministic specifications generally suffice for applications.

Theorem 1. *For all SMTS S, T, $d_t(S,T) \sqsubseteq d_m(S,T)$. If T is deterministic, then $d_t(S,T) = d_m(S,T)$.*

In a quantitative framework, it can be useful to be able to *relax* and *strengthen* specifications during the development process. Which precise relaxations and strengthenings one wishes to apply will depend on the actual application, but we can here show three general relaxations which differ from each other in the *level* of the theory at which they are applied. For $\alpha \in \mathbb{L}$ and SMTS S, T,

- T is an *α-widening* of S if there is a relation $R \subseteq S \times T$ for which $(s_0, t_0) \in R$ and such that for all $(s,t) \in R$, $s \overset{k}{\dashrightarrow}_S s'$ if and only if $t \overset{\ell}{\dashrightarrow}_T t'$, and $s \xrightarrow{k}_S s'$ if and only if $t \xrightarrow{\ell}_T t'$, for $k \sqsubseteq_{\mathsf{Spec}} \ell$, $d(\ell, k) \sqsubseteq \alpha$, and $(s', t') \in R$;
- T is an *α-relaxation* of S if $S \leq_m T$ and $T \leq_m^\alpha S$;
- the *α-extended implementation semantics* of S is $[\![S]\!]^{+\alpha} = \{I \mid \leq_m^\alpha S \mid I \text{ implementation}\}$.

Hence α-widening is an entirely *syntactic* notion: up to unweighted bisimulation, T is the same as S, but transition labels in T can be α "wider" than in S (hence also $S \leq_m T$). The second notion, α-relaxation, works at the level of semantics of specifications, whereas the last notion is at implementation level. A priori, there is no relation between the syntactic and semantic notions, even though one can be established in some special cases.

Example 1 (contd). In [2] it is shown that for the accumulated distance with discounting factor λ, any α-widening is also a $(1-\lambda)^{-1}\alpha$-relaxation. This is due to the fact that for traces $\sigma, \tau \in \mathsf{Spec}^\infty$ with $d(\sigma_j, \tau_j) \leq \alpha$ for all j, we have $\sum_j \lambda^j d(\sigma_j, \tau_j) \leq \sum_j \lambda^j \alpha \leq (1-\lambda)^{-1}\alpha$ by convergence of the geometric series.

Example 2 (contd). For the maximum-lead distance, it is easy to expose cases of α-widenings which are not β-relaxations for any β. One example consists of two one-state SMTS S, T with loops $s_0 \xrightarrow{a,1} s_0$ and $t_0 \xrightarrow{a,[0,2]} t_0$; then T is a 1-widening of S, but $g \circ d_m(T,S) = \infty$.

Proposition 2. *If T is an α-relaxation of S, then $[\![T]\!] \subseteq [\![S]\!]^{+\alpha}$.*

It can be shown for special cases that the inclusion in the proposition is strict [2]; for the proof one only needs the fact that $d_m(I,S) \sqsubseteq d_m(I,T) \oplus d_m(T,S) \sqsubseteq \alpha$ for all $I \in [\![T]\!]$.

Proposition 3. *Let T be an α-relaxation of S and T' an α'-relaxation of S', and let $d_m(S,S') = \beta$. Then $\beta \sqsubseteq d_m(S,T') \oplus \alpha'$, $d_m(S,T') \sqsubseteq \beta$, $\beta \sqsubseteq d_m(T,S')$, and $d_m(T,S') \oplus \alpha \sqsubseteq \beta$.*

4 Structural Composition and Quotient

We now introduce the different operations on SMTS which make up a specification theory. Firstly, we are interested in composing specifications S, S' into a specification $S\|S'$ by synchronizing on shared actions and with interleaving for non-shared actions. Secondly, we need a quotient operator which solves equations of the form $S\|X \equiv T$, that is, the quotient synthesizes the most general specification $T \oslash S$ which describes all SMTS X satisfying the above equation. We first define a partial *label synchronization* operator $\oplus : \mathsf{Spec} \times \mathsf{Spec} \hookrightarrow \mathsf{Spec}$ which satisfies the following conditions:

- For all $k, \ell, k', \ell' \in \mathsf{Spec}$, if $d(k,\ell) \neq \top$ and $d(k',\ell') \neq \top$, then $k \oplus k'$ is defined if and only if $\ell \oplus \ell'$ is defined;
- for all $\ell, \ell' \in \mathsf{Spec}$, $(\exists k \in \mathsf{Spec} : d(k,\ell) \neq \top\ , d(k,\ell') \neq \top\ ) \iff (\exists m \in \mathsf{Spec} : \ell \oplus m, \ell' \oplus m$ are defined).

This operator permits to synchronize labels at transitions which are executed in parallel; the first property ensures that refinements of synchronable labels can synchronize and *vice versa*, and the second relates synchronizability to distances in such a way that two labels have a common quantitative refinement if and only if they have a common synchronization.

Additionally, we must assume that there exists a function $P : \mathbb{L} \times \mathbb{L} \to \mathbb{L}$ which allows us to infer bounds on distances on synchronized labels. We assume that P is monotone in both coordinates, has $P(\bot\ , \bot\ ) = \bot\ ,\ P(\alpha, \top\ ) = P(\top\ , \alpha) = \top$ for all $\alpha \in \mathbb{L}$, and that

$$F(k \oplus k', \ell \oplus \ell', P(\alpha, \alpha')) \sqsubseteq P(F(k,\ell,\alpha), F(k',\ell',\alpha'))$$

for all $k, \ell, k', \ell' \in \mathsf{Spec}$ and $\alpha, \alpha' \in \mathbb{L}$ for which $k \oplus k'$ and $\ell \oplus \ell'$ are defined. Hence $d(k \oplus k', \ell \oplus \ell') \sqsubseteq P(d(k,\ell), d(k',\ell'))$ for all such $k, \ell, k', \ell' \in \mathsf{Spec}$, thus P indeed bounds distances of synchronized labels.

The *structural composition* of two SMTS S and T is the SMTS $S\|T = (S \times T, (s_0, t_0), \dashrightarrow_{S\|T}, \longrightarrow_{S\|T})$ with transitions defined as follows:

$$\frac{s \overset{k}{\dashrightarrow}_S s' \quad t \overset{\ell}{\dashrightarrow}_T t' \quad k \oplus \ell \text{ defined}}{(s,t) \overset{k\oplus\ell}{\dashrightarrow}_{S\|T} (s',t')} \qquad \frac{s \overset{k}{\longrightarrow}_S s' \quad t \overset{\ell}{\longrightarrow}_T t' \quad k \oplus \ell \text{ defined}}{(s,t) \overset{k\oplus\ell}{\longrightarrow}_{S\|T} (s',t')}$$

The next theorem shows that structural composition supports *independent implementability*: if S is close to T and S' close to T', then we can bound the distance between the structural compositions.

Theorem 2. *For SMTS S, T, S', T', $d_m(S\|S', T\|T') \sqsubseteq P(d_m(S,T), d_m(S',T'))$.*

Example 1 (contd). In [2], structural composition of labels is defined by

$$(a,[l,r]) \oplus (a',[l',r']) = \begin{cases} (a,[l+l', r+r']) & \text{if } a = a', \\ \text{undefined} & \text{otherwise.} \end{cases}$$

This composition is bounded above by $P(\alpha, \alpha') = \alpha + \alpha'$, hence Theorem 2 specializes to [2, Thm. 5]: $d_m(S \| S', T \| T') \leq d_m(S, T) + d_m(S', T')$.

Example 2 (contd). For the max-lead distance, and with an application to real-time systems in mind, structural composition is more naturally defined using *intersection* of intervals rather than addition, that is,

$$(a, [l, r]) \oplus (a', [l', r']) = \begin{cases} (a, [\max(l, l'), \min(r, r')]) \\ \qquad\qquad \text{if } a = a' \text{ and } \max(l, l') \leq \min(r, r'), \\ \text{undefined} \qquad \text{otherwise.} \end{cases}$$

(Note that, however, the definition of structural composition is independent of which distance one uses; one might as well combine addition of intervals, as in Example 1, with the max-lead distance.) This composition is bounded by $P(\alpha, \alpha') = \max(\alpha, \alpha')$, thus $d_m(S \| S', T \| T') \leq \max(d_m(S, T) + d_m(S', T'))$.

For *quotients* of SMTS, we need a partial label operator $\oslash : \mathsf{Spec} \times \mathsf{Spec} \to \mathsf{Spec}$ for which it holds that

- for all $k, \ell, m \in \mathsf{Spec}$, $\ell \oslash k$ is defined and $m \sqsubseteq_{\mathsf{Spec}} \ell \oslash k$ if and only if $k \oplus m$ is defined and $k \oplus m \sqsubseteq_{\mathsf{Spec}} \ell$;
- for all $\ell, \ell' \in \mathsf{Spec}$, $(\exists k \in \mathsf{Spec} : d(k, \ell) \neq \top \ , d(k, \ell') \neq \top \ ) \iff (\exists m \in \mathsf{Spec} : m \oslash \ell, m \oslash \ell'$ are defined).

The first condition ensures that $\oslash$ is inverse to $\oplus$, and the second relates it to distances just as we did for $\oplus$ above. We extend the first condition to say that $\oslash$ is *quantitatively well-behaved* if it holds for all $k, \ell, m \in \mathsf{Spec}$ that $\ell \oslash k$ is defined and $d(m, \ell \oslash k) \neq \top$ if and only if $k \oplus m$ is defined and $d(k \oplus m, \ell) \neq \top$, and in that case, $F(m, \ell \oslash k, \alpha) \sqsupseteq F(k \oplus m, \ell, \alpha)$ for all $\alpha \in \mathbb{L}$. We say that $\oslash$ is *quantitatively exact* if the inequality can be sharpened to $F(m, \ell \oslash k, \alpha) = F(k \oplus m, \ell, \alpha)$.

In the definition of quotient below, we denote by $\rho_B(S)$ the *pruning* of a SMTS S with respect to the states in $B \subseteq S$, which is obtained as follows. Define a *must*-predecessor operator $\mathsf{pre} : 2^S \to 2^S$ by $\mathsf{pre}(S') = \{s \in S \mid \exists k \in \mathsf{Spec}, s' \in S' : s \xrightarrow{k} s'\}$ and let pre^* be the reflexive, transitive closure of pre. Then $\rho_B(S)$ exists if $s_0 \notin \mathsf{pre}^*(B)$, and in that case, $\rho_B(S) = (S_\rho, s_0, \dashrightarrow_\rho, \longrightarrow_\rho)$ with $S_\rho = S \backslash \mathsf{pre}^*(B)$, $\dashrightarrow_\rho = \dashrightarrow \cap (S_\rho \times \mathsf{Spec} \times S_\rho)$, and $\longrightarrow_\rho = \longrightarrow \cap (S_\rho \times \mathsf{Spec} \times S_\rho)$.

For SMTS S, T, the *quotient* of T by S is the SMTS $T \setminus\!\!\setminus S = \rho_B(T \times S \cup \{u\}, (t_0, s_0), \dashrightarrow_{T \setminus\!\!\setminus S}, \longrightarrow_{T \setminus\!\!\setminus S})$ given as follows (if it exists):

$$\frac{t \overset{\ell}{\dashrightarrow}_T t' \quad s \overset{k}{\dashrightarrow}_S s' \quad \ell \oslash k \text{ defined}}{(t, s) \overset{\ell \oslash k}{\dashrightarrow}_{T \setminus\!\!\setminus S} (t', s')} \qquad \frac{t \overset{\ell}{\longrightarrow}_T t' \quad s \overset{k}{\longrightarrow}_S s' \quad \ell \oslash k \text{ defined}}{(t, s) \overset{\ell \oslash k}{\longrightarrow}_{T \setminus\!\!\setminus S} (t', s')}$$

$$\frac{t \overset{\ell}{\longrightarrow}_T t' \quad \forall s \overset{k}{\longrightarrow}_S s' : \ell \oslash k \text{ undefined}}{(t, s) \in B}$$

$$\frac{m \in \mathsf{Spec} \quad \forall s \overset{k}{\dashrightarrow}_S s' : k \oplus m \text{ undefined}}{(t, s) \overset{m}{\dashrightarrow}_{T \setminus\!\!\setminus S} u} \qquad \frac{m \in \mathsf{Spec}}{u \overset{m}{\dashrightarrow}_{T \setminus\!\!\setminus S} u}$$

The next theorem shows that under certain standard conditions, quotient is *sound* and *maximal* with respect to $\|$. Note that the property that $X \leq_m T \setminus\!\setminus S$ iff $S\|X \leq_m T$ implies *uniqueness* of quotient [11]; hence if a certain instantiation of our framework admits a quotient which is not quantitatively well-behaved, there is no hope that one can find another one which is.

Theorem 3. *Let S, T, X be SMTS such that S is deterministic and $T \setminus\!\setminus S$ exists. Then $X \leq_m T \setminus\!\setminus S$ if and only if $S\|X \leq_m T$. Also,*

- *if $\oslash$ is quantitatively well-behaved, then $d_m(X, T \setminus\!\setminus S) \sqsupseteq \ d_m(S\|X, T)$;*
- *if $\oslash$ is quantitatively exact and $d_m(X, T \setminus\!\setminus S) \neq \top$, then $d_m(X, T \setminus\!\setminus S) = d_m(S\|X, T)$.*

Example 1 (contd). In [2], quotient of labels is defined by

$$(a', [l', r']) \oslash (a, [l, r]) = \begin{cases} (a, [l' - l, r' - r]) & \text{if } a = a' \text{ and } l' - l \leq r' - r, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

This operator is quantitatively exact, hence Theorem 3 specializes to [2, Thm. 6]: if $d_m(X, T \setminus\!\setminus S) \neq \infty$, then $d_m(X, T \setminus\!\setminus S) = d_m(S\|X, T)$.

Example 2 (contd). For structural composition using interval intersection, it can be shown that $\oslash$ given by

$$(a', [l', r']) \oslash (a, [l, r]) = \begin{cases} \text{undefined} & \text{if } a \neq a', \\ (a, [l', \infty]) & \text{if } a = a' \text{ and } l < l' \leq r \leq r', \\ (a, [l', r']) & \text{if } a = a' \text{ and } l < l' \leq r' < r, \\ \text{undefined} & \text{if } a = a' \text{ and } l \leq r < l' \leq r', \\ (a, [-\infty, \infty]) & \text{if } a = a' \text{ and } l' \leq l \leq r \leq r', \\ (a, [-\infty, r']) & \text{if } a = a' \text{ and } l' \leq l \leq r < r', \\ \text{undefined} & \text{if } a = a' \text{ and } l' \leq r' < l \leq r. \end{cases}$$

yields a quotient operator which is quantitatively well-behaved, but *not* exact. Theorem 3 implies that $d_m(X, T \setminus\!\setminus S) \geq d_m(S\|X, T)$ if if $d_m(X, T \setminus\!\setminus S) \neq \infty$.

5 Conjunction

Conjunction of SMTS can be used to merge two specifications into one. Let $\owedge : \mathsf{Spec} \times \mathsf{Spec} \to \mathsf{Spec}$ be a partial label operator for which it holds that

- for all $k, \ell \in \mathsf{Spec}$, if $k \owedge \ell$ is defined, then $k \owedge \ell \sqsubseteq_{\mathsf{Spec}} k$, $k \owedge \ell \sqsubseteq_{\mathsf{Spec}} \ell$, and
- for all $\ell, \ell' \in \mathsf{Spec}$, $(\exists k \in \mathsf{Spec} : d(k, \ell) \neq \top \ , d(k, \ell') \neq \top \ ) \iff (\exists m \in \mathsf{Spec} : \ell \owedge m, \ell' \owedge m$ are defined).

The first requirement above ensures that conjunction acts as a lower bound, and the second one relates it to distances such that two labels have a common refinement if and only if they have a common conjunction. One also usually

wants conjunction to be a *greatest* lower bound; we say that $\oslash$ is *conjunctively compositional* if it holds for all $k, \ell, m \in \mathsf{Spec}$ for which $m \sqsubseteq_{\mathsf{Spec}} k$ and $m \sqsubseteq_{\mathsf{Spec}} \ell$ that also $k \oslash \ell$ is defined and $m \sqsubseteq_{\mathsf{Spec}} k \oslash \ell$.

As a quantitative generalization, and analogously to what we did for structural composition, we say that $\oslash$ is *conjunctively bounded* by a function $C : \mathbb{L} \times \mathbb{L} \to \mathbb{L}$ if C is monotone in both coordinates, has $C(\bot\ , \bot\ ) = \bot\ $, $C(\alpha, \top\ ) = C(\top\ , \alpha) = \top\ $ for all $\alpha \in \mathbb{L}$, and if it holds for all $k, \ell, m \in \mathsf{Spec}$ for which $d(m, k) \neq \top\ $ and $d(m, \ell) \neq \top\ $ that $k \oslash \ell$ is defined and

$$F(m, k \oslash \ell, C(\alpha, \alpha')) \sqsubseteq\ C(F(m, k, \alpha), F(m, \ell, \alpha'))$$

for all $\alpha, \alpha' \in \mathbb{L}$. Note that this implies that $d(m, k \oslash \ell) \sqsubseteq\ C(d(m, k), d(m, \ell))$, hence conjunctive boundedness implies conjunctive compositionality.

The *conjunction* of two SMTS S and T is the SMTS $S \wedge T = \rho_B(S \times T, (s_0, t_0), \dashrightarrow_{S\wedge T}, \longrightarrow_{S\wedge T})$ given as follows:

$$\frac{s \xrightarrow{k}_S s' \quad t \overset{\ell}{\dashrightarrow}_T t' \quad k \oslash \ell \text{ defined}}{(s,t) \xrightarrow{k\oslash\ell}_{S\wedge T} (s',t')} \qquad \frac{s \overset{k}{\dashrightarrow}_S s' \quad t \xrightarrow{\ell}_T t' \quad k \oslash \ell \text{ defined}}{(s,t) \xrightarrow{k\oslash\ell}_{S\wedge T} (s',t')}$$

$$\frac{s \overset{k}{\dashrightarrow}_S s' \quad t \overset{\ell}{\dashrightarrow}_T t' \quad k \oslash \ell \text{ defined}}{(s,t) \overset{k\oslash\ell}{\dashrightarrow}_{S\wedge T} (s',t')}$$

$$\frac{s \xrightarrow{k}_S s' \quad \forall t \overset{\ell}{\dashrightarrow}_T t' : k \oslash \ell \text{ undef.}}{(s,t) \in B} \qquad \frac{t \xrightarrow{\ell}_T t' \quad \forall s \overset{k}{\dashrightarrow}_S s' : k \oslash \ell \text{ undef.}}{(s,t) \in B}$$

The next theorem shows the precise conditions under which conjunction is a greatest lower bound. Note that the greatest-lower-bound condition $U \leq_m S$, $U \leq_m T \Longrightarrow U \leq_m S \wedge T$ entails uniqueness.

Theorem 4. *Let S, T, U be SMTS. If $S \wedge T$ is defined, then $S \wedge T \leq_m S$ and $S \wedge T \leq_m T$. If, additionally, S or T are deterministic, then:*

- *If $\oslash$ is conjunctively compositional, $U \leq_m S$, and $U \leq_m T$, then $S \wedge T$ is defined and $U \leq_m S \wedge T$.*
- *If $\oslash$ is conjunctively bounded by C, $d_m(U, S) \neq \top\ $, and $d_m(U, T) \neq \top\ $, then $S \wedge T$ is defined and $d_m(U, S \wedge T) \sqsubseteq\ C(d_m(U, S), d_m(U, T))$.*

Example 1 (contd). For the formalism of [2], there is a conjunction operator $\oslash$ given by intersection of intervals:

$$(a, [l, r]) \oslash (a', [l', r']) = \begin{cases} (a, [\max(l, l'), \min(r, r')]) & \text{if } a = a', \max(l, l') \leq \min(r, r'), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

This operator is conjunctively compositional, but not conjunctively bounded, whence [2, Thm. 4]: by uniqueness, there does not exist any bounded conjunction operator within the formalism of [2].

To deal with the problem that, as in Example 1, conjunction may not be conjunctively bounded, we introduce another, weaker, property which ensures some

compatibility of conjunction with distances. We say that $\mathbin{⍉}$ is *relaxed conjunctively bounded* by a function family $C = \{C_{\beta,\gamma} : \mathbb{L} \times \mathbb{L} \to \mathbb{L} \mid \beta, \gamma \in \mathbb{L}\}$ if all $C_{\beta,\gamma}$ are monotone in both coordinates, have $C_{\beta,\gamma}(\bot\ ,\bot\ ) = \bot\ , C_{\beta,\gamma}(\alpha, \top\ ) = C_{\beta,\gamma}(\top\ , \alpha) = \top\ $ for all $\alpha \in \mathbb{L}$, and if it holds for all $k, \ell \in \mathsf{Spec}$ for which there is $m \in \mathsf{Spec}$ with $d(m,k) \neq \top\ $ and $d(m,\ell) \neq \top\ $ that there exist $k', \ell' \in \mathsf{Spec}$ with $k \sqsubseteq_{\mathsf{Spec}} k'$, $\ell \sqsubseteq_{\mathsf{Spec}} \ell'$, $d(k',k) = \beta \neq \top\ $, and $d(\ell',\ell) = \gamma \neq \top\ $, such that $k' \mathbin{⍉} \ell'$ is defined, and then for all $m \in \mathsf{Spec}$ with $d(m,k) \neq \top\ $ and $d(m,\ell) \neq \top\ $,

$$F(m, k' \mathbin{⍉} \ell', C_{\beta,\gamma}(\alpha,\alpha')) \sqsubseteq\ C_{\beta,\gamma}(F(m,k,\alpha), F(m,\ell,\alpha'))$$

for all $\alpha, \alpha' \in \mathbb{L}$. The following theorem shows that relaxed boundedness of $\mathbin{⍉}$ entails a similar property for conjunction.

Theorem 5. *Let S, T be SMTS with S or T deterministic and $\mathbin{⍉}$ relaxed conjunctively bounded by C. If there is an SMTS U for which $d_m(U,S), d_m(U,T) \neq \top\ $, then there exist β- and γ-widenings S' of S and T' of T for which $S' \wedge T'$ is defined, and such that $d_m(U, S' \wedge T') \sqsubseteq\ C_{\beta,\gamma}(d_m(U,S), d_m(U,T))$ for all SMTS U for which $d_m(U,S) \neq \top\ $ and $d_m(U,T) \neq \top\ $.*

Example 1 (contd). For the accumulating distance, $\mathbin{⍉}$ is relaxed conjunctively bounded by $C_{\beta,\gamma}(\alpha,\alpha') = \max(\alpha,\alpha') \oplus\ \max(\beta,\gamma)$. Hence Theorem 5 entails that if S or T are deterministic and there is U for which $d_m(U,S), d_m(U,T) \neq \infty$, then there exist a β-widening S' of S and a γ-widening T' of T for which the conjunction $S' \wedge T'$ is defined, and such that $d_m(U, S' \wedge T') \leq \max(d_m(U,S), d_m(U,T)) + \max(\beta,\gamma)$ for all SMTS U for which $d_m(U,S) \neq \top\ $ and $d_m(U,T) \neq \top\ $.

Example 2 (contd). Also for the max-lead distance, $\mathbin{⍉}$ given by intersection of intervals is the unique conjunction operator. It is again relaxed conjunctively bounded by $C_{\beta,\gamma}(\alpha,\alpha') = \max(\alpha,\alpha') \oplus\ \max(\beta,\gamma)$, hence the same specialization of Theorem 5 as above holds for the max-lead distance.

6 Conclusion

We believe that this paper constitutes the first general and complete quantitative theory for modal specifications. We have shown not only how to introduce such a general quantitative framework, but also the general conditions one needs to impose on the interplay between the system distance and the operators such as composition and quotient for the quantitative theory to work properly.

Using [2] and our running example of max-lead distances, we have seen two different instantiations of the general framework, using different distances for measuring variations of systems and specifications and different operators for structural composition and quotient. Application of our framework *e.g.* to real-time and hybrid systems, in programming languages or quantitative logics, will require other distances and other operators, but as shown in [10,12], they all stay within the unifying framework introduced in this paper.

References

1. Aliprantis, C.D., Border, K.C.: Infinite Dimensional Analysis: A Hitchhiker's Guide. Springer (2007)
2. Bauer, S.S., Fahrenberg, U., Juhl, L., Larsen, K.G., Legay, A., Thrane, C.: Quantitative Refinement for Weighted Modal Transition Systems. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 60–71. Springer, Heidelberg (2011)
3. Bauer, S.S., Juhl, L., Larsen, K.G., Legay, A., Srba, J.: Extending modal transition systems with structured labels. Math. Struct. CS (2012)
4. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N.: Quantitative analysis of real-time systems using priced timed automata. CACM 54(9), 78–87 (2011)
5. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Trans. Comp. Logic 11(4) (2010)
6. David, A., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: Timed I/O automata: A complete specification theory for real-time systems. In: HSCC, pp. 91–100. ACM (2010)
7. de Alfaro, L., Faella, M., Stoelinga, M.: Linear and branching system metrics. IEEE Trans. Soft. Eng. 35(2), 258–273 (2009)
8. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. Int. J. Game Th. 8, 109–113 (1979)
9. Fahrenberg, U., Juhl, L., Larsen, K.G., Srba, J.: Energy Games in Multiweighted Automata. In: Cerone, A., Pihlajasaari, P. (eds.) ICTAC 2011. LNCS, vol. 6916, pp. 95–115. Springer, Heidelberg (2011)
10. Fahrenberg, U., Legay, A., Thrane, C.: The quantitative linear-time–branching-time spectrum. In: FSTTCS. LIPIcs, vol. 13, pp. 103–114 (2011)
11. Fahrenberg, U., Legay, A., Wąsowski, A.: Vision Paper: Make a Difference! (Semantically). In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 490–500. Springer, Heidelberg (2011)
12. Fahrenberg, U., Thrane, C., Larsen, K.G.: Distances for weighted transition systems: Games and properties. In: QAPL. EPTCS, vol. 57, pp. 134–147 (2011)
13. Godefroid, P., Huth, M., Jagadeesan, R.: Abstraction-Based Model Checking Using Modal Transition Systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 426–440. Springer, Heidelberg (2001)
14. Gruler, A., Leucker, M., Scheidemann, K.: Modeling and Model Checking Software Product Lines. In: Barthe, G., de Boer, F.S. (eds.) FMOODS 2008. LNCS, vol. 5051, pp. 113–131. Springer, Heidelberg (2008)
15. Grumberg, O., Lange, M., Leucker, M., Shoham, S.: Don't Know in the μ-Calculus. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 233–249. Springer, Heidelberg (2005)
16. Henzinger, T.A., Majumdar, R., Prabhu, V.S.: Quantifying Similarities Between Timed Systems. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 226–241. Springer, Heidelberg (2005)
17. Juhl, L., Larsen, K.G., Srba, J.: Modal transition systems with weight intervals. J. Logic Alg. Prog. (2012)
18. Larsen, K.G.: Modal Specifications. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990)
19. Larsen, K.G., Fahrenberg, U., Thrane, C.: Metrics for weighted transition systems: Axiomatization and complexity. Th. Comp. Sci. 412(28), 3358–3369 (2011)
20. Munkres, J.R.: Topology. Prentice Hall (2000)

21. Nyman, U.: Modal Transition Systems as the Basis for Interface Theories and Product Lines. PhD thesis, Aalborg University (September 2008)
22. Rasmussen, J.I., Larsen, K.G., Subramani, K.: On using priced timed automata to achieve optimal scheduling. Formal Meth. Syst. Design 29(1), 97–114 (2006)
23. Thrane, C., Fahrenberg, U., Larsen, K.G.: Quantitative simulations of weighted transition systems. J. Logic Alg. Prog. 79(7), 689–703 (2010)
24. van Breugel, F.: A theory of metric labelled transition systems. Annals of the New York Academy of Sciences 806(1), 69–87 (1996)
25. van Breugel, F.: A Behavioural Pseudometric for Metric Labelled Transition Systems. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 141–155. Springer, Heidelberg (2005)
26. Zwick, U., Paterson, M.: The Complexity of Mean Payoff Games. In: Li, M., Du, D.-Z. (eds.) COCOON 1995. LNCS, vol. 959, pp. 1–10. Springer, Heidelberg (1995)

The Complexity of Intersecting Finite Automata Having Few Final States

Michael Blondin and Pierre McKenzie

Département d'informatique et de recherche opérationnelle,
Université de Montréal, Montréal, QC, Canada H3T 1J4
{blondimi,mckenzie}@iro.umontreal.ca

Abstract. The problem of determining whether several finite automata accept a common word is closely related to the well-studied membership problem in transformation monoids. We review the complexity of the intersection problem and raise the issue of limiting the number of final states in the automata involved. In particular, we consider commutative automata with at most two final states and we partially elucidate the complexity of their intersection nonemptiness and related problems.

1 Introduction

Let $[m]$ denote $\{1, 2, ..., m\}$ and let PS be the *point-spread problem* for transformation monoids, which we define as follows:

Input: $m > 0$, $g_1, \ldots, g_k : [m] \to [m]$ and $S_1, \ldots, S_m \subseteq [m]$.
Question: $\exists g \in \langle g_1, \ldots, g_k \rangle$ such that $i^g \in S_i$ for every $i \in [m]$?

Here $\langle g_1, \ldots, g_k \rangle$ denotes the monoid obtained by closing the set $\{\mathsf{id}_m, g_1, \ldots, g_k\}$ under function composition and i^g denotes the image of i under g.

The PS problem generalizes many problems found in the literature. For example, it generalizes the (transformation monoid) membership problem [Koz77] Memb, the pointset transporter problem [LM88] and the set transporter problem [LM88]. Moreover, it largely amounts to none other than the *finite automata nonemptiness intersection problem*, AutoInt, defined as follows:

Input: finite automata $A_1, \ldots, A_k$ on a common alphabet Σ.
Question: $\exists w \in \Sigma^*$ accepted by A_i for every $i \in [k]$?

As we note in Prop. 2.1, PS and AutoInt are indeed equivalent in terms of their complexity, even when the monoid in the PS instances and the transition monoids of the automata in the AutoInt instances are drawn from a fixed monoid variety. We view PS as mildly more fundamental because it involves a single monoid.

Memb and AutoInt were shown to be PSPACE-complete by Kozen [Koz77]. Shortly afterwards, the connection with the graph isomorphism problem led to an in-depth investigation of permutation group problems. In particular, Memb was shown to belong to P for groups [FHL80], then to NC^3 for abelian groups [MC87, Mul87], to NC for nilpotent groups [LM88], solvable groups [LM88],

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 31–42, 2012.

groups with bounded non-abelian composition factors [Luk86], and finally all groups [BLS87]. A similar complexity classification of Memb for group-free (or aperiodic) monoids owes to [Koz77, Bea88a, BMT92], who show that Memb for any fixed aperiodic monoid variety is either in AC^0, in P, in NP, or in PSPACE (and complete for that class with very few exceptions).

On the other hand, AutoInt has received less attention. This is (or might be) due to the fact that it is equivalent to the membership problem when both are intractable, but appears harder to solve than the membership problem when the latter is efficiently solvable. For example, Beaudry [Bea88b] shows AutoInt for abelian groups and AutoInt for idempotent commutative monoids to be NP-complete. Beaudry points out that those two cases are examples where the automata intersection problem seems strictly harder than the transformation monoid membership problem (whose complexity is NC^3 for abelian groups and AC^0 for idempotent commutative monoids). Moreover, early results from [Gal76] show that the problem is NP-complete even when Σ is a singleton.

Nonetheless, interesting results arise from the study of AutoInt. For example, the case where k is bounded by a function in the input length was studied in [LR92]. When $k \leq g(n)$, it is proved that the problem is NSPACE$(g(n) \log n)$-complete under log-space reductions. This arguably provided the first natural complete problems for NSPACE$(\log^c n)$. Moreover, it was proved by Karakostas, Lipton and Viglas that improving the best algorithms known solving AutoInt for a constant number of automata would imply NL $\neq$ P [KLV03].

More recently, the intersection problem was also studied for regular expressions without binary + operators [Bal02], instead of finite automata. It is shown to be PSPACE-complete for expressions of star height 2 and NP-complete for star height (at most) 1. Finally, the parameterized complexity of a variant of the problem, where Σ^c is considered instead of Σ^*, was examined in [War01]. Different parameterizations of c, k and the size of the automata yield FPT, NP, W[1], W[2] and W[t] complexities. More results on AutoInt are surveyed in [HK11].

1.1 Our Contribution

We propose PS as the right algebraic formulation of AutoInt. We observe that PS generalizes known problems and we identify PS variants that are both efficiently solvable and interesting. We obtain these variants by restricting the transition monoids of the automata, or the number of generators (alphabet size), or by bounding the size of the S_i s (number of final states) to less than 3.

We then mainly investigate monoids that are abelian groups, but we also consider groups, commutative monoids and idempotent monoids. In the case of abelian groups, we revisit the equivalences with AGM (abelian permutation group membership) and LCON (feasibility of linear congruences with tiny moduli) [MC87], which have further been investigated recently in the context of log-space counting classes [AV10]. Focussing on the cases involving one or two final states complements Beaudry's hardness proofs for the intersection problem [Bea88b], which require at least three final states. We obtain the partial classification of PS for abelian groups given by Table 1 below.

Table 1. Complexity of the point-spread problem for abelian groups

	Maximum size of S_i for every $i \in [m]$		
	1	2	3+
Single generator	L-complete	L-complete	NP-complete
Elementary 2-groups	$\oplus$L-complete	$\oplus$L-complete	NP-complete [Bea88b]
Elementary p-groups	Mod_pL-complete	? ($\in$ NP)	NP-complete [Bea88b]
General case	NC^3, $\mathrm{FL}^{\mathrm{ModL}}$/poly	? ($\in$ NP)	NP-complete [Bea88b]

The first line of Table 1 gives the complexity of AutoInt for abelian groups with $|\Sigma| = 1$, but it applies also to all automata over $\Sigma = \{a\}$ and to a class of abelian group automata which we will call *tight abelian group automata*. To the best of our knowledge, Table 1 yields the first efficiently solvable variants of the automata intersection problem. Moreover, it provides characterizations of Mod_pL and thus allows the study of (some) log-space counting classes in terms of automata. The case of two final states, although incomplete, is of special interest since it appears to be efficiently solvable and generalizes group theory and linear algebra problems.

We also introduce a generalization of AutoInt by adding $\cup$-clauses. More formally, the problem is to determine whether $\cap_{i=1}^{k} \cup_{j=1}^{k'} \mathrm{Language}(A_{i,j}) \neq \emptyset$. When $k' = 2$ and each automaton possesses one final state, this generalizes the original version of the problem with two final states. In the case of unary languages, we are able to show this variant to be NL-complete and thus suggest this definition to be the right generalization, in-between two and three final states, to avoid complexity blow-ups.

Section 2 presents our notation, defines the relevant problems and relates PS and AutoInt to some algebraic problems. Section 3 is devoted to the analysis of the complexity of PS and AutoInt for abelian group automata subject to multiple restrictions. A short Section 4 contains observations about the complexity of PS and AutoInt in commutative monoids. Section 5 concludes and mentions open problems.

2 Preliminaries

2.1 Basic Definitions and Notation

An *automaton* refers to a deterministic complete finite automaton. Formally, it is a tuple $(\Omega, \Sigma, \delta, \alpha, F)$ where Ω is the set of *states*, Σ is an *alphabet*, $\delta : \Omega \times \Sigma \to \Omega$ is the *transition function*, $\alpha \in \Omega$ is the *initial state* and $F \subseteq \Omega$ is the set of *final states* (*accepting states*). The language of an automaton A is denoted $\mathrm{Language}(A)$. The number of occurrences of σ in a word w is denoted by $|w|_\sigma$. Throughout the paper, the automata always share the same alphabet and we denote its size $|\Sigma|$ by s.

The *transition monoid* $\mathcal{M}(A)$ of an automaton A is the monoid $\langle \{T_\sigma : \sigma \in \Sigma\} \rangle$ where $T_\sigma(\gamma) = \delta(\gamma, \sigma)$. For $w = w_1 \cdots w_\ell$, $T_w = T_{w_\ell} \circ \cdots \circ T_{w_1}$. When $\mathcal{M}(A)$

is a group, and thus a permutation group on Ω, every letter $\sigma \in \Sigma$ has an *order* $\mathrm{ord}(\sigma)$ that may be defined by the order of T_σ in $\mathcal{M}(A)$. However, we prefer considering the automaton A' obtained from removing the states not accessible from the initial state of A. Therefore, we define $\mathrm{ord}(\sigma)$ as the order of T_σ in the transitive permutation group $\mathcal{M}(A')$. For an automaton A, we say that A is an (abelian) group automaton if its transition monoid is an (abelian) group.

An abelian group automaton A will be said to be a *tight abelian group automaton* if $\{v \in \mathbb{Z}_{\mathrm{ord}(\sigma_1)} \times \cdots \times \mathbb{Z}_{\mathrm{ord}(\sigma_s)} : \sigma_1^{v_1} \cdots \sigma_s^{v_s} \in \mathrm{Language}(A)\}$ contains only one element. We note that when $\Sigma = \{a\}$, the automata are directed cycles of size $\mathrm{ord}(a)$, and thus accept only one word of size less than $\mathrm{ord}(a)$. Another family fulfilling this criterion is the set of automata obtained by taking the cartesian product of unary automata working on distinct letters.

Automata are encoded by their transition monoid. We assume any reasonable encoding of monoids, described in terms of their generators, that allows basic operations like composing two transformations and determining the image of a point under a transformation in AC^0. We use the notation $\leq^m_{\log}$ for log-space many-one reductions and $\leq^T_{\mathrm{NC}^1}$ for NC^1 Turing reductions. Equivalences are defined analogously and denoted by $\equiv$. See [MC87] for more details.

A function f is in GapL iff f is logspace many-one reducible to computing the integer determinant of a matrix [ABO99]. A language S is in $\mathrm{Mod}_k\mathrm{L}$ [BDHM92] iff there exists $f \in \#\mathrm{L}$ such that $x \in S \Leftrightarrow f(x) \not\equiv 0 \pmod{k}$. A language S is in ModL [AV10] iff there exists $f \in \mathrm{GapL}, g \in \mathrm{FL}$ such that for all strings x, $g(x) = 0^{p^e}$ for some prime p and $e \in \mathbb{N}$, and $x \in S \Leftrightarrow f(x) \equiv 0 \pmod{|g(x)|}$. For every prime power p^e, $\mathrm{Mod}_{p^e}\mathrm{L} \subseteq \mathrm{ModL} \subseteq \mathrm{NC}^2$, and $\mathrm{FL}^{\mathrm{ModL}} = \mathrm{FL}^{\mathrm{GapL}}$ [AV10].

Let p be a prime. A finite group is a p-group iff its order is a power of p. An abelian group is an abelian elementary p-group iff every non trivial element has order p. A finite group is nilpotent iff it is the direct product of p-groups.

We use lcm for the least common multiple, gcd for the greatest common divisor, n for the input length, and $\mathbb{Z}_q$ for the integers mod q. We say that an integer q is *tiny* if its value is smaller than the input length (i.e. $|q| \leq n$).

2.2 Problems

We define and list the problems mentioned in this paper for ease of reference. Here X is any family of finite monoids, such as all commutative monoids, or all abelian groups, or all groups. In this paper, X will always be a pseudovariety, i.e., a family of finite monoids closed under finite direct products and under taking homomorphic images of submonoids; see [BMT92] for an argument that such families are a rich and natural choice.

$\mathsf{PS}_b(X)$ (Point-spread problem)

Input: $m > 0$, $g_1, \ldots, g_k : [m] \to [m]$ such that $\langle g_1, \ldots, g_k \rangle \in X$, and $S_1, \ldots, S_m \subseteq [m]$, such that $|S_i| \leq b$ or $|S_i| = m$ for every $i \in [m]$.

Question: $\exists g \in \langle g_1, \ldots, g_k \rangle$ such that $i^g \in S_i$ for every $i \in [m]$?

$\mathsf{AutoInt}_b(X)$ (Automata nonemptiness intersection problem)

Input: finite automata $A_1, \ldots, A_k$ on a common alphabet Σ, such that $\mathcal{M}(A_i) \in X$ and A_i has at most b final states for every $i \in [k]$.
Question: $\exists w \in \Sigma^*$ accepted by A_i for every $i \in [k]$?

$\mathsf{Memb}(X)$ (Membership problem)

Input: $m > 0$, $g_1, \ldots, g_k : [m] \to [m]$ such that $\langle g_1, \ldots, g_k \rangle \in X$, and $g : [m] \to [m]$.
Question: $g \in \langle g_1, \ldots, g_k \rangle$?

$\mathsf{PT}(X)$ (Pointset transporter)

Input: $m > 0$, $g_1, \ldots, g_k : [m] \to [m]$ such that $\langle g_1, \ldots, g_k \rangle \in X$, and $b_1, \ldots, b_r \in [m]$ for some $r \leq m$.
Question: $\exists g \in \langle g_1, \ldots, g_k \rangle$ such that $i^g = b_i$ for every $i \in [r]$?

$\mathsf{ST}(X)$ (Set transporter)

Input: $m > 0$, $g_1, \ldots, g_k : [m] \to [m]$ such that $\langle g_1, \ldots, g_k \rangle \in X$, $r \leq m$ and $B \subseteq [m]$.
Question: $\exists g \in \langle g_1, \ldots, g_k \rangle$ such that $\{1^g, 2^g, \ldots, r^g\} \subseteq B$?

LCON (Linear congruences)

Input: $B \in \mathbb{Z}^{k \times l}, b \in \mathbb{Z}^k$, and an integer q presented as a list of its tiny factors $p_1^{e_1}, \ldots, p_r^{e_r}$.
Question: $\exists x \in \mathbb{Z}^l$ satisfying $Bx \equiv b \pmod{q}$?

$\mathsf{LCONNULL}$ (Linear congruences "nullspace")

Input: $B \in \mathbb{Z}^{k \times l}$, and an integer q presented as a list of its tiny factors $p_1^{e_1}, \ldots, p_r^{e_r}$.

Problem: compute a generating set for the $\mathbb{Z}$-module $\{x \in \mathbb{Z}^l : Bx \equiv 0 \pmod{q}\}$.

$\mathsf{PS}(X)$ and $\mathsf{AutoInt}(X)$ refer to $\mathsf{PS}_b(X)$ and $\mathsf{AutoInt}_b(X)$ with no bound placed on b. Moreover, we refer to b as the number of final states, even in the context of PS. When the modulus q is fixed to a constant, we use the notation LCON_q and $\mathsf{LCONNULL}_q$.

The point-spread problem relates to other problems as follows.

Proposition 2.1. $\mathsf{AutoInt}_b(X) \equiv^m_{\mathrm{NC}^1} \mathsf{PS}_b(X)$ *for any finite monoid variety* X.

Proof. $\mathsf{AutoInt}_b(X) \leq^m_{\mathrm{NC}^1} \mathsf{PS}_b(X)$: Let $\Omega = \Omega_1 \cup \cdots \cup \Omega_k$. For each $\sigma \in \Sigma$, let g_σ be the transformation action of the letter σ on Ω. For each $\gamma \in \Omega$, let

$$S_\gamma = \begin{cases} F_i & \text{if } \gamma \text{ is the initial state of } A_i, \\ \Omega & \text{if } \gamma \text{ is any other state of } A_i. \end{cases}$$

Let α_i be the initial state of A_i, then there is a word w accepted by every automaton iff $\alpha_i{}^{g_w} \in F_i$ for every $i \in [k]$ iff g_w maps every initial state to a final state. To complete the reduction, one must notice that $|S_\gamma|$ is either equal to $|\Omega|$ or bounded by b. Moreover, $\langle \{g_\sigma : \sigma \in \Sigma\} \rangle \in X$ since it is a submonoid of $\mathcal{M}(A_1) \times \cdots \times \mathcal{M}(A_k)$.

$\mathsf{PS}_b(X) \leq^m_{\mathrm{NC}^1} \mathsf{AutoInt}_b(X)$: For every $i \in [m]$, let $A_i = ([m], \{g_1, \ldots, g_k\}, \delta, i, S_i)$ where $\delta : [m] \times \{g_1, \ldots, g_k\} \to [m]$ maps (j, g_ℓ) to j^{g_ℓ} for every $j \in [m]$, $\ell \in [k]$. When $S_i = [m]$, we do not build any automaton since it would accept Σ^*. We note that there exists $g \in \langle g_1, \ldots, g_k \rangle$ such that $i^g \in S_i$ for every $i \in [m]$ iff g is accepted by every automaton. Moreover, every automaton has at most b final states and $\mathcal{M}(A_i) \in X$. □

Proposition 2.2. $\mathsf{Memb}(X) \leq^m_{\mathrm{NC}^1} \mathsf{PT}(X) \equiv^m_{\mathrm{NC}^1} \mathsf{PS}_1(X)$ *and* $\mathsf{ST}(X) \leq^m_{\mathrm{NC}^1} \mathsf{PS}(X)$.

Proof. We use the same generators for every reduction. For $\mathsf{Memb}(X) \leq^m_{\mathrm{NC}^1} \mathsf{PT}(X)$, we let $b_i = i^g$ for every $i \in [m]$ where g is the given test transformation. For $\mathsf{PT}(X) \leq^m_{\mathrm{NC}^1} \mathsf{PS}_1(X)$, we let $S_i = \{b_i\}$ for every $i \in [r]$ and $S_i = [m]$ otherwise. For $\mathsf{PS}_1(X) \leq^m_{\mathrm{NC}^1} \mathsf{PT}(X)$, if $|S_i| = 1$, we let b_i be the unique element of S_i. To be consistent with the definition, the points should be reordered such that the points transported come first. Finally, for $\mathsf{ST}(X) \leq^m_{\mathrm{NC}^1} \mathsf{PS}(X)$, we let $S_i = B$ for every $i \in [r]$, and $S_i = [m]$ for every i such that $r < i \leq m$. □

Proposition 2.3. *If* $\mathsf{Memb}(X) \in$ NP (PSPACE) *then* $\mathsf{PS}(X) \in$ NP (PSPACE).

Proof. We guess a transformation g such that $i^g \in S_i$ for $i \in [m]$. From there, we execute the NP (PSPACE) machine for $\mathsf{Memb}(X)$ to test whether $g \in \langle g_1, \ldots, g_k \rangle$. For the PSPACE result, we use PSPACE = NPSPACE [Sav70]. □

3 Groups and Abelian Groups

In this section we consider groups. We first record that $\mathsf{PS}_1(\text{Groups}) \in$ NC, owing to a slick parallel reduction [Luk90, p. 27] from $\mathsf{PT}(X)$ to the problem of computing pointwise stabilizers, also known [BLS87] to be in NC. It follows that $\mathsf{PS}(\text{Groups})$ is in NP by Propositions 2.2 and 2.3, and complete for NP by the forthcoming Theorem 3.12.

Proposition 3.1. $\mathsf{PS}_1(\textit{Groups}) \in$ NC *and* $\mathsf{PS}(\textit{Groups})$ *is* NP-complete.

We have been unable so far to solve $\mathsf{PS}_2(\text{Groups})$. It is shown in [LM88] that $\mathsf{PT}(\text{Nilpotent groups}) \in$ NC, so that $\mathsf{PS}_1(\text{Nilpotent groups}) \in$ NC by Proposition 2.2. This implies that both problems belong to NC for abelian groups.

The rest of our investigation of PS in the group case is devoted to abelian groups. We first refine the above NC upper bound for $\mathsf{PS}_1(\text{Abelian groups})$ to NC^3, namely the same complexity as $\mathsf{Memb}(\text{Abelian groups})$. To achieve this, we show that $\mathsf{AutoInt}_1(\text{Abelian groups}) \leq \mathsf{LCONNULL}$. Such a reduction can be extracted from [MC87]. However, we sketch a direct reduction, not considered explicitly in [MC87], here:

Proposition 3.2 ([MC87]). $\mathsf{AutoInt}_1(\textit{Abelian groups}) \leq^T_{\mathrm{NC}^1} \mathsf{LCONNULL}$.

Proof. Let $\sigma \in \Sigma$ and $\mathrm{ord}_i(\sigma)$ be the order of σ in A_i. Let $\Phi_i = \{v \in \mathbb{Z}_{q_i}^s : T_{\sigma_1^{v_1}\cdots\sigma_s^{v_s}}(\alpha_i) = \alpha_i\}$ where $q_i = \mathrm{lcm}(\mathrm{ord}_i(\sigma_1), \ldots, \mathrm{ord}_i(\sigma_s))$ and α_i is the initial state of A_i. Let B_i be the matrix such that each line is a vector from a generating set for $\Phi_i^\perp = \{v \in \mathbb{Z}_{q_i}^s : \forall u \in \Phi_i,\ u \cdot v = 0\}$, which may be obtained by computing a generating set for Φ_i and then calling an oracle for $\mathsf{LCONNULL}$. A generating set for Φ_i may be computed in logarithmic space because checking the existence of a word $w \in \Sigma^*$ accepted by an abelian group automaton such that $|w|_{\sigma_i} = c_i$ for every $1 \leq i \leq k$ can be done by testing accessiblity in an undirected graph. Let β_i be the final state of A_i, and $b_i = B_i x_i$ where $x_i = (|w_i|_{\sigma_1} \bmod q_i, \ldots, |w_i|_{\sigma_s} \bmod q_i)$ for any word w_i such that $T_{w_i}(\alpha_i) = \beta_i$. Then, there is a word accepted by every automaton iff this instance of LCON is feasible:

$$\begin{pmatrix} B_1 & q_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ B_k & 0 & \cdots & q_k \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_s \\ y_1 \\ \vdots \\ y_k \end{pmatrix} \equiv \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} \pmod{\mathrm{lcm}(q_1, \ldots, q_k)}.$$

□

Since $\mathsf{LCONNULL} \in \mathrm{NC}^3$ [MC87] and $\mathsf{LCONNULL} \in \mathrm{FL}^{\mathrm{ModL}}/\mathrm{poly}$ [AV10], we obtain the following corollaries.

Corollary 3.3. $\mathsf{AutoInt}_1(\textit{Abelian groups})$ *is in* NC^3 *and* $\mathrm{FL}^{\mathrm{ModL}}/\textit{poly}$.

By Proposition 2.2, $\mathsf{Memb}(\text{Abelian groups}) \leq^m_{\mathrm{NC}^1} \mathsf{AutoInt}_1(\text{Abelian groups})$. Since $\mathsf{Memb}(\text{Abelian groups}) \in \mathrm{NC}^3$ [MC87], we obtain a rather tight bound.

We now restrict our abelian groups to elementary abelian p-groups. This allows a characterization of the complexity class $\mathrm{Mod}_p\mathrm{L}$ (denoted $\oplus\mathrm{L}$ when $p = 2$) by the intersection problem, and thus in terms of automata.

Theorem 3.4. $\mathsf{AutoInt}_1(\textit{Elementary abelian } p\textit{-groups})$ *is* $\mathrm{Mod}_p\mathrm{L}$-complete.

Proof (sketch). In an elementary abelian p-group, every (nontrivial) element has order p. Therefore $\mathrm{lcm}(\mathrm{ord}_i(\sigma_1), \ldots, \mathrm{ord}_i(\sigma_s)) = p$ (or 1). Thus, the reduction from Proposition 3.2 may be converted to a log-space computable reduction to $\mathsf{LCONNULL}_p$ without any significant modification. A log-space reduction from LCON_p is also easily obtained by mapping each equation to an automaton. Since LCON_p and $\mathsf{LCONNULL}_p$ are both $\mathrm{Mod}_p\mathrm{L}$-complete [BDHM92], and $\mathrm{Mod}_p\mathrm{L} = \mathrm{Mod}_p\mathrm{L}^{\mathrm{Mod}_p\mathrm{L}}$ ($\mathrm{FMod}_p\mathrm{L} = \mathrm{FL}^{\mathrm{Mod}_p\mathrm{L}}$) [HRV00], we obtain the desired result. □

We now give the first result of this paper concerning the intersection problem with each automaton having two final states. When the transition monoids are restricted to elementary abelian 2-groups, we are able to reduce $\mathsf{AutoInt}_2$ to LCON_2. Therefore, in this case, the problem with two final states per automaton is not harder than with one final state.

Theorem 3.5. $\mathsf{AutoInt}_2$(*Elementary abelian 2-groups*) *is* $\oplus$L-complete.

Proof (sketch). We give a reduction to $\mathsf{LCONNULL}_2$. By following the proof of Proposition 3.2, we can derive a system of linear congruences $(B_i x \equiv b_i \pmod 2) \vee (B_i x \equiv b'_i \pmod 2)$ for every $i \in [k]$, feasible if and only if there is a word accepted by every automaton. The $\vee$-clauses are removed by introducing additional variables. More formally, we build $B_i x \equiv z_i b_i + z'_i b'_i \pmod 2$ for every $i \in [k]$ with constraints $z_i + z'_i \equiv 1 \pmod 2$ forcing the selection of either b_i or b'_i. □

We may now study the case where Σ consists of a single letter a. Instead of directly considering unary automata, we study the more general case of tight abelian group automata. Before proceeding, we note that the intersection problem over unary languages in general is not harder. Indeed, an automaton over a singleton alphabet consists of a tail and a cycle. Words accepted by the tail of an automaton may be tested first on the whole collection. If none is accepted, the associated final states are removed and an equivalent cyclic automaton is built.

We first consider a generalization of $\mathsf{AutoInt}$, denoted $\mathsf{AutoInt}(\cup^{k'})$, that consists of determining whether $\cap_{i=1}^{k} \cup_{j=1}^{k'} \text{Language}(A_{i,j}) \neq \emptyset$. We examine the case of $\mathsf{AutoInt}_1(\cup^2)$ that generalizes $\mathsf{AutoInt}_2$, and show it is NL-complete for unary and tight abelian group automata.

We will use the following generalization of the Chinese remainder theorem:

Lemma 3.6. *[Knu81, see p. 277 ex. 3] Let* $a_1, \ldots, a_k \in \mathbb{N}$ *and* $q_1, \ldots, q_k \in \mathbb{N}$. *There exists* $x \in \mathbb{N}$ *such that* $x \equiv a_i \pmod{q_i}$ *for every* $i \in [k]$ *iff* $a_i \equiv a_j \pmod{\gcd(q_i, q_j)}$ *for every* $i, j \in [k]$.

Theorem 3.7. $\mathsf{AutoInt}_1(\bigcup^2$ *Tight abelian group automata*$) \leq^m_{\log}$ 2–SAT.

Proof. Let $A[i,0]$ and $A[i,1]$ be the two automata of the i^{th} $\cup$-clause. Let $v[i,x]$ be the unique vector of $V[i,x] = \{v \in \mathbb{Z}_{\text{ord}_{i,x}(\sigma_1)} \times \cdots \times \mathbb{Z}_{\text{ord}_{i,x}(\sigma_s)} : \sigma_1^{v_1} \cdots \sigma_s^{v_s} \in \text{Language}(A[i,x])\}$ which is computable in log-space. We first note that $A[i,x]$ accepts exactly words $w \in \Sigma^*$ such that $|w|_{\sigma_j} \equiv v[i,x]_j \pmod{\text{ord}_{i,x}(\sigma_j)}$ for every $j \in [s]$, by definition of $V[i,x]$. Therefore, distinct letters are independent and we may find a word accepted by every automaton by verifying restrictions locally on $\sigma_1, \ldots, \sigma_s$. Thus, we have the following equivalences:

$$
\begin{aligned}
&\exists w \text{ such that } w \in \bigcap_{i=1}^{k} \bigcup_{x=0}^{1} \text{Language}(A[i,x]) \\
\Leftrightarrow\ &\exists w\, \exists x \in \{0,1\}^k \text{ such that } w \in \bigcap_{i=1}^{k} \text{Language}(A[i,x_i]) \\
\Leftrightarrow\ &\exists w\, \exists x \in \{0,1\}^k \text{ such that } \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{s} |w|_{\sigma_j} \equiv v[i,x_i]_j \pmod{\text{ord}_{i,x_i}(\sigma_j)}
\end{aligned}
$$

$$\Leftrightarrow \exists w\, \exists x \in \{0,1\}^k \text{ such that } \bigwedge_{j=1}^{s} \left(\bigwedge_{i=1}^{k} |w|_{\sigma_j} \equiv v[i,x_i]_j \pmod{\mathrm{ord}_{i,x_i}(\sigma_j)} \right)$$

$$\Leftrightarrow \exists x \in \{0,1\}^k \text{ such that } \bigwedge_{j=1}^{s} \left(\bigwedge_{i=1}^{k} \bigwedge_{i'=1}^{k} C_{i,i',j}(x) \right),$$

where $C_{i,i',j}(x) = \big(v[i,x_i]_j \equiv v[i',x_{i'}]_j \pmod{\gcd(\mathrm{ord}_{i,x_i}(\sigma_j), \mathrm{ord}_{i',x_{i'}}(\sigma_j))}\big)$.

The last equivalence is a consequence of Lemma 3.6. Therefore, there is a word accepted by every automaton iff this last Boolean expression is satisfiable. For every $i,i' \in [k], j \in [s]$, the truth table of $C_{i,i',j}$ may be computed by evaluating the four congruences. Since $C_{i,i',j}$ depends only on two variables, it is always possible to obtain a 2-CNF expression. Moreover, the congruences are computable in logarithmic space since the numbers implied are tiny. □

Theorem 3.8. 2–SAT $\leq^m_{\log}$ AutoInt$_1(\bigcup^2$ *Abelian groups with* $|\Sigma| = 1)$.

Proof. Let $C(x)$ be the Boolean expression $\bigwedge_{i=1}^{k} C_i(x)$ over $x_1, \ldots, x_m$ where $C_i(x) = (x_{r_i} \oplus b_i) \vee (x_{t_i} \oplus b'_i)$ and $b_i, b'_i \in \{0,1\}$ indicate whether negation must be taken or not.

It is possible to represent an assignment with an integer, assuming it is congruent to 0 or 1 mod the m first primes $p_1, \ldots, p_m$. The remainder of such an integer mod p_i represents the value of the i^{th} variable. Let

$$E_j = \{w \in \{a\}^* : |w| \equiv 0 \pmod{p_j} \vee |w| \equiv 1 \pmod{p_j}\},$$
$$X_i = \{w \in \{a\}^* : |w| \equiv \neg b_i \pmod{p_{r_i}} \vee |w| \equiv \neg b'_i \pmod{p_{t_i}}\}.$$

The language $E_1 \cap \cdots \cap E_m$ represents valid assignments and X_i represents assignments satisfying C_i (but may contain invalid assignments, i.e. not congruent to 0 or 1). The language E_j (resp. X_i) is recognized by the union of two cyclic automata of size p_j (resp. size p_{r_i} and p_{t_i}). It remains to point out that $(E_1 \cap \cdots \cap E_m) \cap (X_1 \cap \cdots \cap X_k) \neq \emptyset$ iff C is satisfiable. □

Corollary 3.9. AutoInt$_1(\bigcup^2$ *Tight abelian group automata*$)$ *and* AutoInt$_1(\bigcup^2$ *Abelian groups with* $|\Sigma| = 1)$ *are* NL-complete.

Recall, that 2–⊕SAT is defined similarly to 2–SAT but with ⊕ operators instead of ∨. It is SL-complete [JLL76] and thus L-complete by SL = L [Rei05].

Theorem 3.10. AutoInt$_2$(*Tight abelian group automata*) $\leq^m_{\log}$ 2–⊕SAT.

Proof. We first note that an automaton with two final states may be replaced with the union of two copies of the same automaton, each having one final state. Thus, we may use the proof of Theorem 3.7. However, it remains to show that it is possible to build an expression in 2-⊕CNF (instead of 2-CNF).

To achieve this, we first note that each letter σ_j has the same order in $A[i,0]$ and $A[i,1]$ (according to Theorem 3.7 notation). We denote this common order by $\mathrm{ord}_i(\sigma_j)$. Therefore, there is a word accepted by every automaton iff $\bigwedge_{j=1}^{s} \bigwedge_{i=1}^{k} \bigwedge_{i'=1}^{k} C_{i,i',j}(x)$ is satisfiable, where

$$C_{i,i',j}(x) = (v[i,x_i]_j \equiv v[i',x_{i'}]_j \pmod{\gcd(\mathrm{ord}_i(\sigma_j), \mathrm{ord}_{i'}(\sigma_j))}).$$

The truth table of $C_{i,i',j}$ may be computed as before by evaluating the four congruences. However, in this case, the modulus is independent of x. Thus, it can be shown that if three of these congruences are true, then all four are. Therefore, $C_{i,i',j}$ can be written solely with the operators $\oplus$ and $\wedge$. □

Corollary 3.11. $\mathsf{AutoInt}_2$(*Tight abelian group automata*) *and* $\mathsf{AutoInt}_2$(*Abelian groups with* $|\Sigma| = 1$) *are* L-complete.

To complete the classification of the intersection problem over unary languages, we argue that it is NP-complete for three final states. A reduction from Monotone 1–in–3 3–SAT [GJ79] may be obtained in a similar fashion to Theorem 3.8. For each clause $(x_1 \vee x_2 \vee x_3)$ we build an automaton with $p_1p_2p_3$ states (and three final states) accepting words $w \in \{a\}^*$ such that

$$(|w| \bmod p_1, |w| \bmod p_2, |w| \bmod p_3) \in \{(1,0,0),(0,1,0),(0,0,1)\}.$$

Theorem 3.12. $\mathsf{AutoInt}_3$(*Tight abelian group automata*) *and* $\mathsf{AutoInt}_3$(*Abelian groups with* $|\Sigma| = 1$) *are* NP-complete.

4 Some Observations on Commutative Monoids

Here we briefly examine the PS problem for monoids (instead of groups). Recall that a monoid is idempotent iff $x^2 = x$ holds for every element x. We first notice that both PS(Idempotent monoids) and PS(Commutative monoids) are NP-complete. This follows from Propositions 2.2 and 2.3, since their Memb counterparts are NP-complete [Bea88a, Bea88b, BMT92].

Proposition 4.1 ([Bea88a, Bea88b, BMT92]). PS(*Idempotent monoids*) *and* PS(*Commutative monoids*) *are* NP-complete*, even for one final state.*

The point-spread problem becomes efficiently solvable when restricted to the variety $\mathbf{J_1}$ of idempotent commutative monoids.

Theorem 4.2. $\mathsf{PS}_1(\mathbf{J_1}) \in \mathrm{AC}^0$.

Proof. We use the technique of [BMT92], for solving $\mathsf{Memb}(\mathbf{J_1})$, based on the so-called maximal alphabet of a transformation. However, we have to be careful since we are dealing with a partially defined transformation. Let $G = \{g_1, \ldots, g_k\}$ and let b_i be the unique element of S_i. Let $A = \{g \in G : {b_i}^g = b_i \quad \forall i \in [r]\}$ and $a = \prod_{g \in A} g$. Suppose there exists $f \in \langle G \rangle$ such that $i^f = b_i$ for every $i \in [r]$. We first notice that $i^{af} = i^f$ for every $i \in [r]$. Indeed, $i^{af} = i^{fa} = b_i^a = b_i = i^f$. Moreover, we have $h_j \in A$ for any h_j appearing in $f = h_1 \cdots h_l$, since $b_i^{h_j} = i^{fh_j} = i^f = b_i$ for every $i \in [r]$. Thus, $i^{af} = i^{a(h_1 \cdots h_l)} = i^a$ for every $i \in [r]$. Therefore $i^a = i^{af} = i^f = b_i$ for every $i \in [r]$. We conclude that there exists $f \in \langle G \rangle$ such that $i^f = b_i$ for every $i \in [r]$ iff $i^a = b_i$ for every $i \in [r]$. This last test can be carried out easily. □

We note that the complexity of $\mathsf{PS}(\mathbf{J_1})$ rises at least to L for two final states.

Proposition 4.3. $\mathsf{PS}_2(\mathbf{J_1})$ *is hard for* L.

Proof. We give a reduction from 2–⊕SAT. For a clause $(x_1 \oplus x_2)$, we build an automaton with four states (and two final states) accepting the language $(\Sigma \setminus \{x_1, x_2\})^*(x_1x_1^* + x_2x_2^*)(\Sigma \setminus \{x_1, x_2\})^*$. For a clause $(x_1 \oplus x_2 \oplus 1)$, we build the automaton accepting the complement of the previous language. □

Unfortunately, we were not able to show $\mathsf{PS}_2(\mathbf{J_1}) \in$ L, even though it appears to be a reasonable claim. The previous hardness proof yields very specific automata, more exactly they have four states and at most one transition between any two distinct states. Only with such an outrageous restriction are we currently able to show a log-space upper bound.

For the sake of completeness, we note that the problem is NP-complete for three final states, as proved in [Bea88b].

Proposition 4.4 ([Bea88b]). $\mathsf{PS}_3(\mathbf{J_1})$ *is* NP-complete.

5 Conclusion and Further Work

The question marks in Table 1 and the lack of upper bound in Proposition 4.3 indicate that further work is needed to properly locate the complexity of the point-spread problem for two final states. Judging from our results for unary languages and elementary abelian 2-groups, we might suspect PS_2 and $\mathsf{AutoInt}_2$ to remain efficiently solvable for abelian permutation groups and for idempotent commutative monoids. However, our current proof for elementary abelian 2-groups cannot be extended. We suspect that PS_2(Elementary abelian p-groups) is harder for $p > 2$ than for $p = 2$.

It would be interesting to answer such questions since PS_2(Abelian groups) is equivalent to testing feasibility of linear congruences of the type $(B_1x \equiv b_1 \pmod{q_1} \vee B_1x \equiv b_1' \pmod{q_1}) \wedge \cdots \wedge (B_kx \equiv b_k \pmod{q_k} \vee B_kx \equiv b_k' \pmod{q_k})$. The variant of $\mathsf{AutoInt}_1$, with two automata per ∪-clause, yields similar linear congruences, but with B_i and q_i differing in each ∨-clause. Therefore, exploring $\mathsf{AutoInt}_2$ and $\mathsf{AutoInt}_2(\cup^2)$ appears to be a fruitful line of research to obtain interesting variants of PS efficiently solvable.

Finally, PS_1(Solvable groups) ∈ NC using [Luk90, p. 27] and [BLS87]. But what about PS_2(Solvable groups) and PS_2(Groups)? An NC toolkit is available, but are the tools sufficient?

References

[ABO99] Allender, E., Beals, R., Ogihara, M.: The complexity of matrix rank and feasible systems of linear equations. Comput. Complex. 8(2), 99–126 (1999)

[AV10] Arvind, V., Vijayaraghavan, T.C.: Classifying problems on linear congruences and abelian permutation groups using logspace counting classes. Comput. Complex. 19, 57–98 (2010)

[Bal02] Bala, S.: Intersection of Regular Languages and Star Hierarchy. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 159–169. Springer, Heidelberg (2002)

[BDHM92] Buntrock, G., Damm, C., Hertrampf, U., Meinel, C.: Structure and importance of logspace-mod class. Theory of Computing Systems 25, 223–237 (1992)

[Bea88a] Beaudry, M.: Membership testing in commutative transformation semigroups. Information and Computation 79(1), 84–93 (1988)

[Bea88b] Beaudry, M.: Membership testing in transformation monoids. PhD thesis, McGill University (1988)

[BLS87] Babai, L., Luks, E.M., Seress, A.: Permutation groups in NC. In: Proc. 19th Annual ACM Symposium on Theory of Computing, pp. 409–420 (1987)

[BMT92] Beaudry, M., McKenzie, P., Thérien, D.: The membership problem in aperiodic transformation monoids. J. ACM 39(3), 599–616 (1992)

[FHL80] Furst, M.L., Hopcroft, J.E., Luks, E.M.: Polynomial-time algorithms for permutation groups. In: FOCS, pp. 36–41 (1980)

[Gal76] Galil, Z.: Hierarchies of complete problems. Acta Informatica 6, 77–88 (1976)

[GJ79] Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman and Company (1979)

[HK11] Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata – a survey. Inf. Comput. 209(3), 456–470 (2011)

[HRV00] Hertrampf, U., Reith, S., Vollmer, H.: A note on closure properties of logspace mod classes. Inf. Process. Lett. 75, 91–93 (2000)

[JLL76] Jones, N.D., Lien, Y.E., Laaser, W.T.: New problems complete for nondeterministic log space. Theory of Computing Systems 10, 1–17 (1976)

[KLV03] Karakostas, G., Lipton, R.J., Viglas, A.: On the complexity of intersecting finite state automata and NL versus NP. Theoretical Computer Science 302(1-3), 257–274 (2003)

[Knu81] Knuth, D.E.: The art of computer programming: seminumerical algorithms, 2nd edn., vol. 2. Addison-Wesley (1981)

[Koz77] Kozen, D.: Lower bounds for natural proof systems. In: Proc. 18th Annual Symposium on Foundations of Computer Science, pp. 254–266 (1977)

[LM88] Luks, E.M., McKenzie, P.: Parallel algorithms for solvable permutation groups. J. Comput. Syst. Sci. 37(1), 39–62 (1988)

[LR92] Lange, K.-J., Rossmanith, P.: The Emptiness Problem for Intersections of Regular Languages. In: Havel, I.M., Koubek, V. (eds.) MFCS 1992. LNCS, vol. 629, pp. 346–354. Springer, Heidelberg (1992)

[Luk86] Luks, E.M.: Parallel algorithms for permutation groups and graph isomorphism. In: FOCS, pp. 292–302. IEEE Computer Society (1986)

[Luk90] Luks, E.M.: Lectures on polynomial-time computation in groups. Technical report. College of Computer Science, Northeastern University (1990)

[MC87] McKenzie, P., Cook, S.A.: The parallel complexity of abelian permutation group problems. SIAM J. Comput. 16, 880–909 (1987)

[Mul87] Mulmuley, K.: A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. Combinatorica 7, 101–104 (1987)

[Rei05] Reingold, O.: Undirected st-connectivity in log-space. In: Proc. 37th Annual ACM Symposium on Theory of Computing, pp. 376–385 (2005)

[Sav70] Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. Syst. Sci. 4(2), 177–192 (1970)

[War01] Wareham, H.T.: The Parameterized Complexity of Intersection and Composition Operations on Sets of Finite-State Automata. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 302–310. Springer, Heidelberg (2001)

News about Semiantichains and Unichain Coverings*

Bartłomiej Bosek[1], Stefan Felsner[2], Kolja Knauer[2], and Grzegorz Matecki[1]

[1] Theoretical Computer Science Department
Faculty of Mathematics and Computer Science Jagiellonian University
{bosek,matecki}@tcs.uj.edu.pl
[2] Diskrete Mathematik Institut für Mathematik Technische Universität Berlin
{felsner,knauer}@math.tu-berlin.de

Abstract. We study a min-max relation conjectured by Saks and West: For any two posets P and Q the size of a maximum semiantichain and the size of a minimum unichain covering in the product $P \times Q$ are equal. As a positive result we state conditions on P and Q that imply the min-max relation. However, we also have an example showing that in general the min-max relation is false. This disproves the Saks-West conjecture.

1 Introduction

Partial order theory plays an important role in many disciplines of computer science and engineering. It has applications in distributed computing, concurrency theory, programming language semantics and data mining. Posets and particularly products of posets are used for modeling dynamic behavior of complex systems that can be captured by causal relations. Min-max relations in posets, such as Dilworth's Theorem [1], relate to flow problems, perfect matchings and integer programming. Often the proofs are constructive using methods of combinatorial optimization.

This paper is about min-max relations with respect to chains and antichains in posets. In a poset, *chains* and *antichains* are sets of pairwise comparable and pairwise incomparable elements, respectively. By the *height* $h(P)$ and the *width* $w(P)$ of poset P we mean the size of a largest chain and a largest antichain, respectively.

Dilworth [1] proved that any poset P can be covered with $w(P)$ chains. Greene and Kleitman [4] generalized Dilworth's Theorem. A *k-family* in P is a subset of P that contains no chain with $k+1$ elements. A greedy argument implies that equivalently k-families are those subsets that decompose into k disjoint antichains. We denote the size of a maximal k-family of P by $d_k(P)$ or simply d_k if the poset is unambiguous from the context. The theorem of Greene and Kleitman says that for every k there is a chain-partition $\mathcal{C}$ of P such that $d_k(P) =$

* The first and the fourth author were supported by Polish MNiSW grant N206 4923 38. The second and the third author were partially supported by DFG grant FE-340/7-2 and ESF EuroGIGA project GraDR.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 43–51, 2012.

$\sum_{C\in\mathcal{C}} \min(k, |C|)$. In [9] (also see [12]) Saks proves the theorem of Greene and Kleitman by showing the following equivalent statement.

Theorem 1. *In a product $C \times Q$ where C is a chain, the size of a minimum chain covering with chains of the form $\{c\} \times C'$ and $C \times \{q\}$ equals the size of a maximum subset $S \subseteq C \times Q$ containing no two elements $(c_1, q_1) < (c_2, q_2)$ such that $c_1 = c_2$ or $q_1 = q_2$. In particular this number is $d_{\min(|C|,h(Q))}(Q)$.*

The *Saks-West Conjecture* states a generalization of Theorem 1. In a product $P \times Q$ we call a chain a *unichain* if it is of the form $\{p\} \times C'$ or $C \times \{q\}$. A *semiantichain* is a set $S \subseteq P \times Q$ such that no two distinct elements of S form a unichain. The conjecture states that the size of a largest semiantichain equals the size of a smallest unichain covering. Several partial results and special cases for posets satisfying the conjecture were obtained in [8,10,15,16,17].

This paper is structured as follows. In Section 2 we provide a sufficient criterion for pairs of posets to satisfy the Saks-West Conjecture. This allows to reproduce several known results and to contribute new classes satisfying the conjecture. Moreover, we present a new class of posets, such that all P from this class satisfy the conjecture with any Q. However, in Section 3 we provide a counterexample to the Saks-West Conjecture. The example can be modified to produce an arbitrary large gap between the size of a largest semiantichain and the size of a smallest unichain covering.

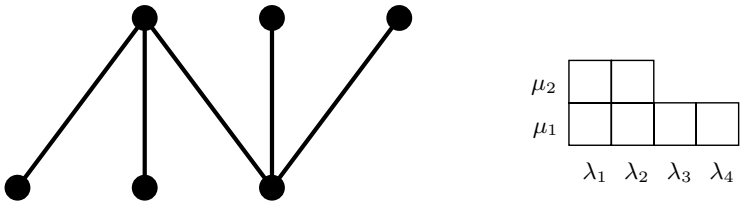
This motivates questions considering complexity issues. How hard are the optimization problems of determining the size of a largest semiantichain or smallest unichain covering for a given $P \times Q$? What is the complexity of deciding whether $P \times Q$ satisfies the Saks-West Conjecture?

2 Constructions

In this section we obtain positive results for posets admitting certain chain and antichain partitions. Dually to the concept of k-family we call a subset of P a k-cofamily if it has width at mos k. Dilworth's Theorem implies that it is thus the union of k disjoint chains. Similarly to $d_k(P)$ we denote the size of a maximal k-cofamily of P by $c_k(P)$ or simply c_k. We will make use of the following theorem of Greene [3]:

Theorem 2. *For any poset P there exists a partition $\lambda^P = \{\lambda_1^P \geq \ldots \geq \lambda_w^P\}$ of $|P|$ such that $c_k(P) = \lambda_1^P + \ldots + \lambda_k^P$ and $d_k(P) = \mu_1^P + \ldots + \mu_k^P$ for each k, where μ^P denotes the partition conjugate to λ^P, i.e., $\mu_i^P = \max\{j \mid \lambda_j^P \geq i\}$ for $i = 1, \ldots, h(P)$.*

Following Viennot [13] we call the Ferrers diagram of λ^P the *Greene diagram* of P, denoted by $G(P)$. We say that P is *d-decomposable* if it has an antichain partition $A_1, A_2, \ldots, A_h$ with $|\bigcup_{i=1}^{k} A_i| = d_k$ for each k. This is, $|A_k| = \mu_k^P$ for all k. In the literature, such an antichain partition is called *completely saturated antichain partition.*



P $G(P)$

Fig. 1. A poset P with its Greene diagram $G(P)$. Note that P is not d-decomposable but c-decomposable.

For posets P and Q with families of disjoint antichains $\{A_1, \ldots, A_k\}$ and $\{B_1, \ldots, B_\ell\}$, respectively, the set $A_1 \times B_1 \cup \ldots \cup A_{\min(k,\ell)} \times B_{\min(k,\ell)}$ is a semiantichain of $P \times Q$. A semiantichain that can be obtained this way is called *decomposable semiantichain*, see [16], where also the following observation was made.

Observation 3. *If P and Q are d-decomposable, then $P \times Q$ has a decomposable semiantichain of size*

$$\sum_{i=1}^{\min(h_P, h_Q)} \mu_i^P \mu_i^Q.$$

In order to construct unichain coverings for $P \times Q$ one can apply Theorem 1 repeatedly. The resulting coverings are called *quasi-decomposable* in [16]. More precisely

Proposition 4. *In a product $P \times Q$ where $\mathcal{C}$ is a chain covering of P there is a unichain covering of size*

$$\sum_{C \in \mathcal{C}} d_{\min(|C|, h(Q))}(Q).$$

Proof. Use Theorem 1 on every $C \times Q$ for $C \in \mathcal{C}$. The union of the resulting unichain coverings is a unichain covering of $P \times Q$ and is of the claimed size. □

Dually to d-decomposable we call P *c-decomposable* if it has a chain partition $C_1, C_2, \ldots, C_w$ with $|\bigcup_{i=1}^{k} C_i| = c_k$, i.e., $|C_k| = \lambda_k^P$ for all k. Chain partitions with the latter property have been referred to as *completely saturated chain partition*, see [16,7]. The following theorem has already been noted implicitly by Tovey and West in [16]. For our proof, we need one more definition: Given posets P and Q with Greene diagrams $G(P)$ and $G(Q)$, respectively, the *merge* of $G(P)$ and $G(Q)$ denoted by $G(P, Q)$ is the set of unit-boxes at coordinates (i, j, k) with $j \leq w(P)$, $i \leq \min(\lambda_j^P, h(Q))$, and $k \leq \mu_i^Q$. Note that the merge is not symmetric with respect to P and Q, as also witnessed by the example in Figure 2.

Theorem 5. *If P is c- and d-decomposable and Q is d-decomposable, then the size of a maximum semiantichain and the size of a minimum unichain covering in the product $P \times Q$ are equal. The size of these is obtained by the two above constructions, i.e.,*

$$\sum_{i=1}^{\min(h_P,h_Q)} \mu_i^P \mu_i^Q = \sum_{j=1}^{w(P)} d_{\min(\lambda_j^P,h(Q))}(Q).$$

Proof. Since P and Q are d-decomposable, there is a semiantichain of size $\sum_{i=1}^{\min(h_P,h_Q)} \mu_i^P \mu_i^Q$ by Observation 3. On the other hand if we take a chain covering $\mathcal{C}$ of P witnessing that P is c-decomposable we obtain a unichain covering of size $\sum_{j=1}^{w(P)} d_{\min(\lambda_j^P,h(Q))}(Q)$ by Proposition 4. We have to prove that these values coincide. Therefore we consider the merge $G(P,Q)$ of the Greene diagrams of P and Q. Counting the boxes in $G(P,Q)$ by j-slices is just the right hand side of our formula. On the other hand if we count the boxes by i-slices we obtain the left hand side of the formula. This concludes the proof. □

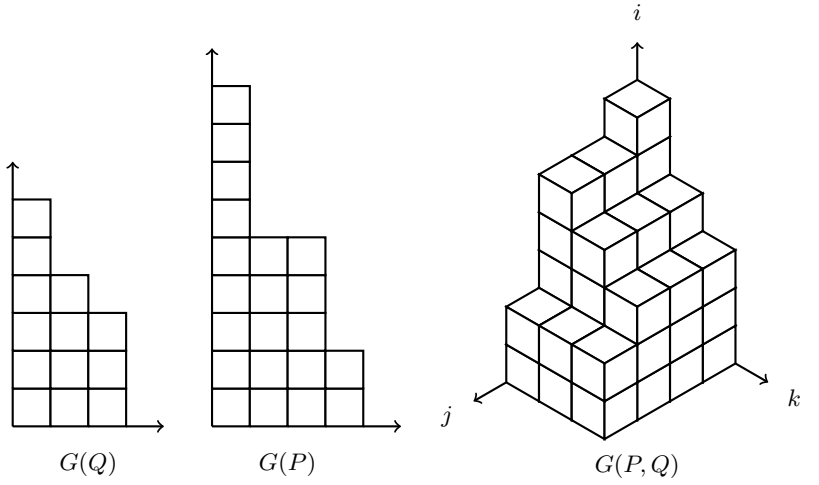


Fig. 2. The merge of two Greene diagrams

Theorem 5 includes some interesting cases for the min-max relation that have been known (•) but also adds a few new cases (⋆). These instances follow from proofs that certain classes of posets are d-decomposable, respectively c-decomposable.

A graded poset P whose ranks yield an antichain partition witnessing that P is d-decomposable is called *strongly Sperner*, see [6]. For emphasis we repeat

- Strongly Sperner posets are d-decomposable.
  For a chain C in P denote by $r(C)$ the set of ranks used by C. A chain-partition $\mathcal{C}$ of P is called *nested* if for each $C, C' \in \mathcal{C}$ we have $r(C) \subseteq r(C')$ if $|C| \leq |C'|$. The most famous class of nested chain partitions are the symmetric chain partitions. In [6] it is proved that nested chain-partitions are completely saturated and that posets admitting a nested chain partition are strongly Sperner. Hence we have the following

- Posets that have a nested chain partition are c- and d-decomposable.
  The fact that products of posets with nested chain partitions satisfy the Saks-West Conjecture was proved earlier in [15]. A special class of strongly Sperner posets are *LYM posets*. A conjecture of Griggs [5] that remains open [2,14] and seems interesting in our context is that LYM posets have completely saturated chain-partitions, i.e., are c-decomposable.

⋆ Orders of width at most 3 are d-decomposable.

Proof. Since P has width at most 3 there are only three possible values of μ_i in the Greene diagram of P. Let a, b, c be the numbers of 3s, 2s, and 1s in $\mu_1, \ldots, \mu_k$, respectively. We have to find an antichain partition of P such that a antichains will be of size 3, b antichains will be of size 2 and c antichains will have size 1. Let $A \subseteq P$ be a maximum $(a+b)$-family. We have the following $|A| = 3a + 2b$, $h(A) = a + b$ and $|P - A| = c$. The last gives us c antichains of size 1. To find the other antichains consider a partition $A = \bigcup_{i=1}^{h(A)} B_i$ such that B_i is the set of minimal points in $B_i \cup \ldots \cup B_{h(A)}$. Since $|B_i| \leqslant 3$ we may consider a', b', c' as the numbers of 3s, 2s, and 1s in all $|B_i|$ (for $i = 1, \ldots, h(A)$). With these numbers we have $|A| = 3a' + 2b' + c'$ and $h(A) = a' + b' + c'$. Obviously, $a' \leqslant a$ because otherwise there would be an $(a+1)$-antichain in A bigger than $3a+2$. Also, note that $|A| - 2h(A) = a = a' - c'$. Thus $c' = 0$, $a' = a$ and $b' = b$. This concludes the proof. □

⋆ Series-parallel orders are c- and d-decomposable.

Proof. This can be proved by induction on the size of the poset. Therefore let the antichain-partitions $\{A_1, \ldots, A_{h(P)}\}$ and $\{A'_1, \ldots, A'_{h(P')}\}$ be witnesses for d-decomposability of P and P', respectively. In a parallel composition $P \parallel P'$ any k-family decomposes into a k-family of P and one of P' and vice-versa. Hence $d_k(P \parallel P') = d_k(P) + d_k(P')$. Say $h(P) \leq h(P')$, then $\{A_1 \cup A'_1, \ldots, A_{h(P)} \cup A'_{h(P)}, A'_{h(P)+1}, \ldots A'_{h(P')}\}$ is an antichain partition of $P \parallel P'$ proving it to be d-decomposable. In a series composition $P; P'$ any k-family decomposes into a $(k-\ell)$-family of P and an ℓ-family of P' and vice-versa. Ordering $A_1, \ldots, A_{h(P)}, A'_1, \ldots, A'_{h(P')}$ by decreasing size yields a witness for d-decomposability of $P; P'$. The proof for c-decomposability goes along the same lines. □

Pairs of weak orders satisfy the Saks-West Conjecture, which follows directly from a theorem in [16]. Since weak orders form a subclass of series-parallel orders we immediately obtain this result again:

- Weak orders are c- and d-decomposable.
  We call a poset P *rectangular* if P contains a poset L consisting of the disjoint union of w chains of length h and P is contained in a weak order U of height h with levels of size w each. Here containment is meant as an inclusion among binary relations. Clearly, rectangular posets are graded with nested chain-decomposition. Thus they are c-and d-decomposable, but more can be said. According to our knowledge the next result is the strongest generalization of Theorem 1 that has been obtained so far.

Theorem 6. *In a product $P \times Q$ where P is rectangular of width w and height h the size of a largest semiantichain equals the size of a smallest unichain covering. Moreover, this number is $w \cdot d_{\min(h,h(Q))}(Q)$.*

Proof. P contains a poset L consisting of the disjoint union of w chains of length h and P is contained in a weak order U of height h with levels of size w. By Proposition 4 we have a unichain covering of $L \times Q$ of size $\sum_{i=1}^{w} d_{\min(h,h(Q))}(Q)$. Moreover this is an upper bound on the size of a minimum unichain covering of $P \times Q$. On the other hand in $U \times Q$ we can find a decomposable semiantichain as a product of the ranks of U with the antichain decomposition $B_1, \ldots, B_{\min(h,h(Q))}$ of a maximal $\min(h, h(Q))$-family in Q. The size of this semiantichain is then $\sum_{i=1}^{\min(h,h(Q))} w|B_i| = w \cdot d_{\min(h,h(Q))}(Q)$ in $U \times Q$. This is a lower bound on the size of the largest semiantichain in $P \times Q$. This concludes the proof. □

3 A Bad Example

To analyze the upcoming example we need the following property of weak orders.

Proposition 7. *If P is an arbitrary poset and Q is a weak order, then the maximum size of a semiantichain in $P \times Q$ can be expressed as $\sum_{i=1}^{k} |A_i| \cdot \mu_i^P \cdot$ where $A_1, A_2, \ldots, A_k$ is a family of disjoint antichains in P.*

Proof. Let S be a semiantichain in $P \times Q$. For any $X \subseteq Q$ denote by $S(X) := \{p \in P \mid \exists q \in X, (p, q) \in S\}$. Recall that for any $q \in Q$ the set $S(\{q\})$ (or shortly $S(q)$) is an antichain in P. Now take a level $B_i = \{q_1, \ldots, q_k\}$ of Q and let A_i be a maximum antichain among $S(q_1), \ldots, S(p_q)$. Replacing $\{q_1\} \times S(q_1), \ldots, \{q_k\} \times S(q_k)$ in S by $A_i \times B_i$ we obtain S' with $|S'| \geq |S|$. Moreover, since Q is a weak order the $S(B_i)$ are mutually disjoint. This remains true in S'. Thus S' is a semiantichain. Applying this operation level by level we construct a decomposable semiantichain of the desired size. □

Let P and Q be the posets shown on in Figure 3. Since Q is a weak-order we can use Proposition 7 to determine the size of a maximum semiantichain in $P \times Q$ as $15 = 5 \cdot 2 + 5 \cdot 1 = 6 \cdot 2 + 3 \cdot 1$. We focus on two of the maximum semiantichains:

$$S_1 = \{1, 2, 3, 4, 9, 10\} \times \{b, c\} \cup \{6, 7, 8\} \times \{a\}$$

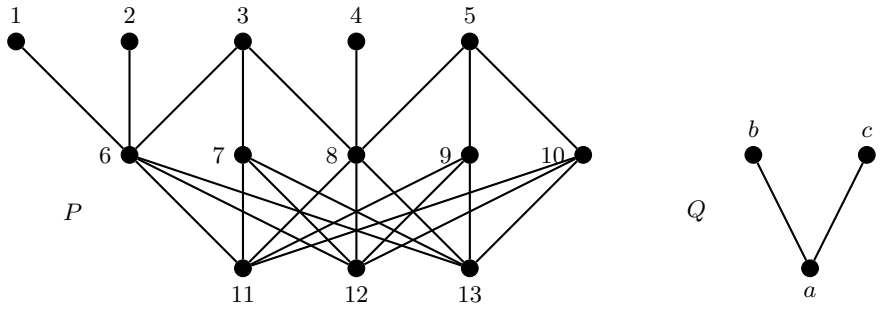


Fig. 3. A pair (P, Q) of posets disproving the conjecture

and

$$S_2 = \{1, 2, 7, 8, 9, 10\} \times \{b, c\} \cup \{3, 4, 5\} \times \{a\}$$

If there is an optimal unichain covering of size 15 then every unichain has to contain one element of each of the two semiantichains S_1 and S_2. We say that the element of a maximum semiantichain is *paying* for the unichain that contains it. Call a unichain with constant Q-component a P-chain and vice versa. Now look at the three points $(11, a), (12, a)$ and $(13, a)$, they have to be covered with P-chains, because a Q-chain containing any of these points cannot be extended to intersect S_1 and S_2. To pay for these three P-chains we need the elements $(6, a), (7, a)$ and $(8, a)$ from S_1 and the elements $(3, a), (4, a)$ and $(5, a)$ from S_2. This implies that $(6, a), (7, a)$ and $(8, a)$ and $(3, a), (4, a)$ and $(5, a)$ have to be covered with the same three chains of the unichain covering. Since the poset induced by these 6 points has width 4 we have reached a contradiction. Consequently there is no unichain covering of $P \times Q$ with 15 unichains.

The above construction can be improved to get the gap between a maximum semiantichain and a minimum unichain covering as big as we want. To see that just replace Q by a height 2 weak order Q' with a set A_1 of k minima and a set A_2 of $k + 1$ maxima. Now consider P' arising from P by blowing up the antichains $\{1, 2\}$ and $\{9, 10\}$ to antichains of size $k + 1$. As in the above proof the size of the maximum semiantichains of $P' \times Q'$ can be determined to be $(k + 4)(k + 1) + (k + 4)k = (2k + 4)(k + 1) + 3k$. As above, the chains containing the $3k$ elements $\{11, 12, 13\} \times A_1$ have to cover the subposet induced by $\{3, 4, 5, 6, 7, 8\} \times A_1$. The latter is of width $4k$. Hence the size of a minimum unichain covering is of size $(2k + 4)(k + 1) + 4k$. We have:

Remark 1. The gap between the size of a maximum semiantichain and a minimum unichain covering in $P' \times Q'$ is k.

Recall that there is no gap if one factor of the product is rectangular (see Theorem 6). Here Q' is almost rectangular but the gap is large.

In [8] some partial results concerning the conjecture were obtained for the class of two-dimensional posets. Note that the two factors in our counterexample are two-dimensional.

Remark 2. In [11] L. E. Trotter and D. B. West define a *uniantichain* to be an antichain in $P \times Q$ in which one of the coordinates is fixed, and define a *semichain* to be a collection of elements of $P \times Q$ in which pairs of elements are comparable if they agree in either coordinate. In [11] it is shown that the size of a minimum semichain covering equals the size of a largest uniantichain. They state the open problem whether the size of a minimum uniantichain covering always equals the size of a largest semichain. We remark that taking the two-dimensional conjugates of P and Q from the previous section yields a negative answer to that question. Furthermore our positive results may be "dualized" to provide classes of posets where the size of a minimum uniantichain covering always equals the size of a largest semichain.

Acknowledgments. We are grateful to Tom Trotter for introducing us to the Saks-West Conjecture and for many fruitful discussions.

References

1. Dilworth, R.P.: A decomposition theorem for partially ordered sets. Ann. of Math. 51(2), 161–166 (1950)
2. Escamilla, E., Nicolae, A., Salerno, P., Shahriari, S., Tirrell, J.: On nested chain decompositions of normalized matching posets of rank 3. Order 28, 357–373 (2011)
3. Greene, C.: Some partitions associated with a partially ordered set. J. Combinatorial Theory Ser. A 20(1), 69–79 (1976)
4. Greene, C., Kleitman, D.J.: The structure of Sperner k-families. J. Combinatorial Theory Ser. A 20(1), 41–68 (1976)
5. Griggs, J.R.: Sufficient conditions for a symmetric chain order. SIAM J. Appl. Math. 32(4), 807–809 (1977)
6. Griggs, J.R.: On chains and Sperner k-families in ranked posets. J. Combin. Theory Ser. A 28(2), 156–168 (1980)
7. Griggs, J.R.: Matchings, cutsets, and chain partitions in graded posets. Discrete Math. 144(1-3), 33–46 (1995); Combinatorics of ordered sets (Oberwolfach, 1991)
8. Liu, Q., West, D.B.: Duality for semiantichains and unichain coverings in products of special posets. Order 25(4), 359–367 (2008)
9. Saks, M.: A short proof of the existence of k-saturated partitions of partially ordered sets. Adv. in Math. 33(3), 207–211 (1979)
10. Tovey, C.A., West, D.B.: Networks and chain coverings in partial orders and their products. Order 2(1), 49–60 (1985)
11. Trotter Jr., L.E., West, D.B.: Two easy duality theorems for product partial orders. Discrete Appl. Math. 16(3), 283–286 (1987)
12. Trotter, W.T.: Partially ordered sets. In: Handbook of Combinatorics, vol. 1, pp. 433–480. Elsevier, Amsterdam (1995)
13. Viennot, X.G.: Chain and antichain families, grids and Young tableaux. In: Orders: Description and Roles (L'Arbresle 1982). North-Holland Math. Stud., vol. 99, pp. 409–463. North-Holland, Amsterdam (1984)

14. Wang, Y.: Nested chain partitions of LYM posets. Discrete Appl. Math. 145(3), 493–497 (2005)
15. West, D.B.: Unichain coverings in partial orders with the nested saturation property. Discrete Math. 63(2-3), 297–303 (1987); Special issue: ordered sets (Oberwolfach, 1985)
16. West, D.B., Tovey, C.A.: Semiantichains and unichain coverings in direct products of partial orders. SIAM J. Algebraic Discrete Methods 2(3), 295–305 (1981)
17. Wu, C.: On relationships between semi-antichains and unichain coverings in discrete mathematics. Chinese Quart. J. Math. 13(2), 44–48 (1998)

Checking Tests for Read-Once Functions over Arbitrary Bases

Dmitry V. Chistikov

Faculty of Computational Mathematics and Cybernetics
Moscow State University, Russia
dch@cs.msu.ru

Abstract. A Boolean function is called read-once over a basis B if it can be expressed by a formula over B where no variable appears more than once. A checking test for a read-once function f over B depending on all its variables is a set of input vectors distinguishing f from all other read-once functions of the same variables. We show that every read-once function f over B has a checking test containing $O(n^l)$ vectors, where n is the number of relevant variables of f and l is the largest arity of functions in B. For some functions, this bound cannot be improved by more than a constant factor. The employed technique involves reconstructing f from its l-variable projections and provides a stronger form of Kuznetsov's classic theorem on read-once representations.

1 Introduction

Let B be an arbitrary set of Boolean functions. A function f is called *read-once* over B iff it can be expressed by a formula over B where no variable appears more than once.

Let $f(x_1, \ldots, x_n)$ be a read-once function over B that depends on all its variables. Then a set M of n-bit vectors is called a *checking test* for f iff for any other read-once function $g(x_1, \ldots, x_n)$ over B there exists a vector $\alpha \in M$ such that $f(\alpha) \neq g(\alpha)$. In other words, M is a checking test for f iff the values of f on vectors from M distinguish f from all other read-once functions g of the same variables. Note that all these *alternatives* g, unlike the *target function* f, may have irrelevant variables.

Denote by B_l the basis of all l-variable Boolean functions. The goal of this paper is to prove that all n-variable read-once functions over B_l have checking tests containing $O(n^l)$ vectors. More generally, for an arbitrary basis B and a read-once function f over B, denote by $T_B(f)$ the smallest possible number of vectors in a checking test for f. This value can be regarded as the *checking test complexity* of f. We show that

$$T_{B_l}\big(f(x_1, \ldots, x_n)\big) \leq 2^l \cdot \binom{n}{l}$$

and, therefore, for any finite basis B and any sequence of read-once functions f_n of n variables over B, it holds that $T_B(f_n) = O(n^l)$ as $n \to \infty$, where l is the largest arity of functions from B.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 52–63, 2012.

This result is based on the previously known relevance hypercube method by Voronenko [15]. Our main contribution is the proof that the method is *correct* for all bases B_l for an arbitrary l, i. e., it provides a way to construct checking tests of specified *length* (cardinality) for all read-once functions over these bases. Previous results give proofs only for $l \leq 5$ [14,15,18].

It should be pointed out that the bound $T_{B_l}(f) = O(n^{l+1})$ can be extracted from the related paper on the exact identification problem by Bshouty, Hancock and Hellerstein [4]. Our result has the following advantages. Firstly, for some functions the bound $O(n^l)$ cannot be improved by more than a constant factor (it matches the known lower bound $\Omega(n^l)$ for n-ary disjunction up to a constant factor). Secondly, checking tests constructed by the relevance hypercube method have regular structure. In short, we show that every read-once function can be unambiguously reconstructed from a set of its l-variable projections with certain properties. This fact may look natural at first sight, but turns out a tricky thing to prove after all.

2 Background and Related Work

Our result has some interesting consequences related to computational learning theory. For instance, it is known that checking tests can be used to implement *equivalence queries* from Angluin's learning model [2]. For the problem of identifying an unknown read-once function over an arbitrary finite basis B with queries, it turns out that non-standard *subcube identity queries* can be efficiently used [7]. A subcube identity query basically asks whether a specified projection of the unknown function f is constant, i. e., whether a given partial assignment of constants to input variables unambiguously determines the value of f.

It follows from our results that for any finite basis B, the problem of learning an unknown read-once function over B can be solved by an algorithm making $O(n^{l+2})$ membership and subcube identity queries, which is polynomial in n (here l is the largest arity of functions in B and a standard *membership query* is simply a request for the value of f on a given input vector). This result builds upon an algorithm by Bshouty, Hancock and Hellerstein [4], which is a strong generalization of a classic exact identification algorithm by Angluin, Hellerstein and Karpinski [3].

Closely related to the notion of checking test complexity is the definition of teaching dimension introduced by Goldman and Kearns [12]. A *teaching sequence* for a Boolean concept (a Boolean function f) in a known class $\mathcal{C}$ is a sequence of labeled instances (pairs of the form $\langle \alpha, f(\alpha) \rangle$) consistent with only one function f from $\mathcal{C}$. The *teaching dimension* of a class is the smallest number t such that all concepts in the class have teaching sequences of length at most t.

In test theory, which dates back to 1950s [6], the corresponding definition is that of the *Shannon function for test complexity*, which is the largest test complexity of an n-variable function. In these terms, our Corollary can be restated as follows: $T_{B_l}(n) = O(n^l)$, where $T_{B_l}(n)$ is the Shannon function for checking test complexity of read-once Boolean functions (i. e., the maximum of $T_{B_l}(f)$

over all n-variable read-once functions over B_l). It must be stressed that in the definition of a checking test used in this paper, the target function is required to depend on all its variables. (One needs to test all 2^n input vectors to distinguish the Boolean constant 0 from all read-once conjunctions of n literals.)

Another appealing problem is that of obtaining bounds on the value of $T_B(f)$ for individual read-once functions f. The bound $T_{B_l}(x_1 \vee \ldots \vee x_n) = \Theta(n^l)$ is obtained in [15] and generalized in [16]. In [8], it is shown that for a wide class of bases including B_l, $l \geq 2$, there exist pairs of read-once functions f, f' such that f' is obtained from f by substituting a constant for a variable and $T_B(f') > T_B(f)$. This result shows that lower bounds on $T_B(f)$ cannot generally be obtained by simply finding projections of f that are already known to require a large number of vectors in their checking tests.

For the basis B_2, individual bounds on the checking test complexity are obtained in [17]. In [19], it is shown that almost all read-once functions over the basis $\{\vee, \oplus\}$ have a relatively small checking test complexity of $O(n \log n)$, as compared to the maximum of $\Theta(n^2)$ (even if alternatives are arbitrary read-once functions over B_2 and not necessarily read-once over $\{\vee, \oplus\}$). For the standard basis $\{\wedge, \vee, \neg\}$, it is known that $n+1 \leq T_{\{\wedge,\vee,\neg\}}(n) \leq 2n+1$ [10], and individual bounds can be deduced from those for the monotone basis $\{\wedge, \vee\}$ [5,9].

3 Basic Definitions

A variable x_i of a Boolean function $f(x_1, \ldots, x_n)$ is called *relevant* (or essential) if there exist two n-bit vectors α and β differing only in the ith component such that $f(\alpha) \neq f(\beta)$. If x_i is relevant to f, then f is said to *depend* on x_i.

In this paper, we call a pair of functions $f(x_1, \ldots, x_n)$ and $g(y_1, \ldots, y_n)$ *similar* if for some constants $\sigma, \sigma_1, \ldots, \sigma_n \in \{0, 1\}$ and for some permutation π of $\{1, \ldots, n\}$ the following equality holds:

$$f(x_1, \ldots, x_n) \equiv g^{\sigma}\left(x_{\pi(1)}^{\sigma_1}, \ldots, x_{\pi(n)}^{\sigma_n}\right),$$

where z^{τ} stands for z if $\tau = 1$ and for $\overline{z}$ if $\tau = 0$. A Boolean function $f(x_1, \ldots, x_n)$, $n \geq 3$, is called *prime* if it has no decomposition of the form

$$f(x_1, \ldots, x_n) \equiv g\big(h(x_{\pi(1)}, \ldots, x_{\pi(k)}),\, x_{\pi(k+1)}, \ldots, x_{\pi(n)}\big),$$

where $1 < k < n$ and π is a permutation of $\{1, \ldots, n\}$.

The structure of formulae expressing read-once functions can be represented by rooted trees. A *tree* of a read-once function $f(x_1, \ldots, x_n)$ over B_l has n leaves labeled with literals of different variables and one or more internal nodes labeled with functions from B_l and symbols $\circ \in \{\wedge, \vee, \oplus\}$ of arbitrary arity (possibly exceeding l). We assume without loss of generality that such trees also have the following properties:

1) any internal node is labeled either with a prime function or with a symbol $\circ \in \{\wedge, \vee, \oplus\}$;

2) internal nodes labeled with identical symbols $\circ \in \{\wedge, \vee, \oplus\}$ are not adjacent.

One can readily see that every read-once function over B_l has at least one tree of this form.

In the sequel, variables are usually identified with corresponding leaves in the tree. Denote by $\mathrm{lca}(y_1, \ldots, y_m)$ the *least common ancestor* of variables $y_1, \ldots, y_m$, i. e., the last common node in (simple) paths from the root of the tree to $y_1, \ldots, y_m$.

Suppose that T is a tree of a read-once function and v is its internal node. By T_v we denote the *subtree of* T *rooted at* v, i. e., the rooted tree that has root v and contains all descendants of v. If $w_1, \ldots, w_p$ are *children* (direct descendants) of v, then subtrees $T_{w_1}, \ldots, T_{w_p}$ are called *subtrees of the node* v. If x is a leaf of T contained in T_v, then by T_v^x we denote a (unique) subtree T_{w_j} containing x. Finally, subtrees of the root node of a tree are called *root subtrees*.

4 The Relevance Hypercube Method

This section is devoted to the review of the relevance hypercube method proposed by Voronenko in [15]. This method has been known to be correct for the bases B_l if $l \leq 5$ (see [15,18]).

From now on, we will use the term "read-once function" instead of "read-once function over B_l". We use boldface letters to denote vectors (often treated as sets) of variables.

Let f be a read-once function depending on variables $\mathbf{x} = \{x_1, \ldots, x_n\}$. Suppose that H is a set of 2^l input vectors disagreeing at most in i_1th, …, i_lth components such that the restriction of f to H (which is an l-variable Boolean function) depends on all its l variables $\mathbf{x}' = \{x_{i_1}, \ldots, x_{i_l}\}$. Then H is called a *relevance hypercube* (or an essentiality hypercube) of dimension l for these variables $\mathbf{x}'$. Any relevance hypercube can be identified with a partial assignment p of constants to input variables such that the induced projection f_p depends on all its l variables. Such assignments are called l-justifying in [4].

Remark. For some functions f and some subsets of their variables relevance hypercubes do not exist. For instance, one may easily check that the function $d(x, u_0, u_1) = (\overline{x} \wedge u_0) \vee (x \wedge u_1)$ has no relevance hypercubes for the set $\mathbf{u} = \{u_0, u_1\}$. As indicated below, the absence of relevance hypercubes is a major obstacle to proving the correctness of relevance hypercube method (see also [15,18]). At the same time, for some functions there exist subsets of variables with more than one relevance hypercube. An example is given by the same function d and the set $\mathbf{u}' = \{x, u_0\}$.

Any set M of input vectors is called a *relevance hypercube set* of dimension l for f if it contains a relevance hypercube H for every l-sized subset of $\mathbf{x}$, for which such a hypercube exists. (Recall that $\mathbf{x}$ is the set of all variables relevant to f.) In other words: consider all l-sized subsets $\mathbf{w} \subseteq \mathbf{x}$ such that f has a relevance hypercube for $\mathbf{w}$. A set M is a relevance hypercube set iff M contains at least

one relevance hypercube for each subset $\mathbf{w}$ of this kind. It was conjectured that any such set is a checking test for f.

Suppose that f is a read-once function that depends on n variables $\mathbf{x}$. Construct a *relevance table* with $\binom{n}{l}$ rows and two columns by the following rule. First, fill the first cells of all rows with different l-sized subsets of $\mathbf{x}$. Then for each row, if the first cell contains a subset $\mathbf{w} \subseteq \mathbf{x}$, put in the second cell any relevance hypercube for $\mathbf{w}$ (along with the corresponding values of f) if such a hypercube exists, or the symbol $*$ otherwise.

In [15], it is shown that any (correct) relevance table uniquely determines a read-once function in the following sense. Suppose that one knows that a function g is read-once and agrees with f on all relevance hypercubes from a relevance table E for f. If one also knows that for each $*$-row in E the function g does not have a relevance hypercube, then one can recursively reconstruct the *skeleton* of g (which is a tree T' such that negating some of its nodes' labels yields a correct tree representing g). After that, the values of f on the vectors from E allow one to prove that g is equal to f.

It follows that a relevance hypercube set M of dimension l for a read-once function f is indeed a checking test for f if E contains no $*$-rows. If for some l-sized subset of variables $\mathbf{w}$ no relevance hypercube exists, then a more sophisticated technique is needed to prove that f can still be reconstructed from its values on vectors from M. The approach used in this paper is outlined in the following Section 5.

5 Assumptions and Notation

In this section we make preliminary assumptions and introduce some notation. All subsequent work, including the proof of our main theorem, is based on the material presented here.

We start with an arbitrary read-once function f over B_l, where $l \geq 3$. Let $\mathbf{x}$ be the set of variables relevant to f. Suppose that M is a relevance hypercube set of dimension l for f. Our goal is to prove that M is a checking test for f, i. e., for any other read-once function $g(\mathbf{x})$ there exists a vector $\alpha \in M$ such that $f(\alpha) \neq g(\alpha)$.

Take any read-once function $g(\mathbf{x})$ that agrees with $f(\mathbf{x})$ on all vectors from M. Firstly and most importantly, we need to prove that root nodes of these two functions' trees are labeled with similar functions. If either of the root nodes is labeled with a symbol $\circ \in \{\wedge, \vee, \oplus\}$, then this can be done with the aid of techniques similar to those from [15]. Here we focus on the prime case, i. e, we assume that

$$f(\mathbf{x}) = f^0(f_1(\mathbf{x}^1), \ldots, f_s(\mathbf{x}^s)),$$
$$g(\mathbf{y}) = g^0(g_1(\mathbf{y}^1), \ldots, g_r(\mathbf{y}^r)),$$

where both f^0 and g^0 are prime, and $\mathbf{x}^1 \cup \ldots \cup \mathbf{x}^s$ and $\mathbf{y}^1 \cup \ldots \cup \mathbf{y}^r$ are partitions of $\mathbf{x} = \mathbf{y}$. (Technically, we must first assume that $\mathbf{y} \subseteq \mathbf{x}$, but it is easily shown

that no variable from $\mathbf{x}$ can be irrelevant to g; see, e. g., Proposition 2 in the next section.) Note that here $s \leq l$ and $r \leq l$.

Suppose we have already proved that f^0 and g^0 are similar. As our second step, we need to show that partitions of input variables into subtrees are identical in the representations above. In other words, we need to show that each $\mathbf{y}^k$ is equal to some $\mathbf{x}^i$.

These two steps, especially the first one, constitute the main difficulties in proving the correctness of the method. The remaining part is technical and can be done with the aid of induction on the depth of the tree representing f. A short explanation of how this part is done is given at the end of our main theorem's proof in Section 8.

In the following sections, we will need the *colouring* of input variables $\mathbf{x}$ defined by the following rule. To each variable $x \in \mathbf{x}$ we assign a (unique) colour $k \in \{1, \ldots, r\}$ such that $x \in \mathbf{y}^k$. This definition provides a convenient way of relating functions f and g (i. e., their tree structure) to each other.

6 Some Observations

In this section we present three facts needed for the sequel. A key observation is given by the following proposition.

Proposition 1. *Suppose that g' is a projection of g that depends on variables x and y having the same colour k. Also suppose that* $\mathrm{lca}(x, y) = v$ *in a tree T' of g'. Then, if v is labeled with a prime function h, it follows that all leaves in the subtree $(T')_v$ have the same colour k. Otherwise, if v is labeled with a symbol $\circ \in \{\wedge, \vee, \oplus\}$, it follows that all leaves in subtrees $(T')_v^x$ and $(T')_v^y$ have the same colour k.*

Proposition 1 follows from a simple fact that substitutions of constants for variables of g can result in removing nodes and subtrees from T', or in replacing nodes with trees that represent projections of prime functions. Adjacent nodes labeled with identical symbols $\circ \in \{\wedge, \vee, \oplus\}$ are subsequently glued together, but least common ancestors of each $\mathbf{y}^i$ either remain roots of single-coloured subtrees, or "support" subsets of single-coloured subtrees of internal nodes labeled with $\circ \in \{\wedge, \vee, \oplus\}$.

For technical reasons, we will also need the following proposition, which holds true for all discrete functions (not necessarily read-once or even Boolean) and follows from Theorem B in [11].

Proposition 2. *Suppose that f is an arbitrary function depending on n variables $\mathbf{x}$. Also suppose that there exists a relevance hypercube for some p variables $\mathbf{u} \subseteq \mathbf{x}$. Then for every q such that $p \leq q \leq n$ there exists a relevance hypercube for some q-sized set of variables $\mathbf{w}$ such that $\mathbf{u} \subseteq \mathbf{w} \subseteq \mathbf{x}$.*

Last but not least, we will use the following fact (see, e. g., [15]).

Proposition 3. *Suppose that a read-once function f is represented by a tree and p variables $\mathbf{u}$ are taken from p different subtrees of an internal node v, which is labeled with a prime function of arity p or with a symbol $\circ \in \{\wedge, \vee, \oplus\}$. Then f has at least one relevance hypercube for $\mathbf{u}$ and, moreover, restrictions of f to all such hypercubes are:*

(*a*) *similar to $h(z_1, \ldots, z_p)$ if v is labeled with a prime function h;*
(*b*) *similar to $z_1 \circ \ldots \circ z_p$ if v is labeled with a symbol $\circ \in \{\wedge, \vee, \oplus\}$.*

7 Auxiliary Lemmas

Suppose that read-once functions f and g satisfy all the assumptions made in Section 5 and T is a tree of f. Recall that by $\mathbf{x}$ we denote the set of all variables relevant to f. We say that a set $\mathbf{u} \subseteq \mathbf{x}$ is *stable* iff for any set $\mathbf{w}$ such that $\mathbf{u} \subseteq \mathbf{w} \subseteq \mathbf{x}$ and any relevance hypercube H for $\mathbf{w}$ there exists a relevance hypercube H' for $\mathbf{u}$ such that $H' \subseteq H$.

Remark. The definition of a stable set $\mathbf{u}$ does not require the existence of relevance hypercubes for all sets $\mathbf{w}$ such that $\mathbf{u} \subseteq \mathbf{w} \subseteq \mathbf{x}$. What is says is that *if* there exists such a relevance hypercube H, then there exists a relevance hypercube for $\mathbf{u}$ which is a subcube of H. For $\mathbf{w} = \mathbf{x}$, however, the definition requires that at least one relevance hypercube for $\mathbf{u}$ exists.

One can readily observe that all singleton subsets of $\mathbf{x}$ are stable. Examples of sets that are not stable are given by $\mathbf{u} = \{u_0, u_1\}$ (as witnessed by $\mathbf{w} = \mathbf{u} \cup \{y\}$) for functions $f_1 = (\overline{x} \wedge d(y, u_0, u_1)) \vee (x \wedge (y \vee u_0 \vee u_1))$ and $f_2 = (\overline{x} \wedge d(y, u_0, u_1)) \vee (x \wedge (u_0 \vee u_1))$, where $d(y, u_0, u_1) = (\overline{y} \wedge u_0) \vee (y \wedge u_1)$. Our main ingredient in the proof of the main theorem is given by the following lemma, which allows us to establish a link between the tree structure of our functions f and g.

Lemma 1. *For any stable set $\mathbf{u}$ of at most l variables of the function f, the function g agrees with f on some relevance hypercube for $\mathbf{u}$.*

Proof. We start with the definition of a stable set. First choose $\mathbf{w} = \mathbf{x}$ and conclude that f has a relevance hypercube for $\mathbf{u}$. By Proposition 2, f also has a relevance hypercube for some l-sized set of variables $\mathbf{w}'$ such that $\mathbf{u} \subseteq \mathbf{w}'$. It follows that M contains some relevance hypercube for $\mathbf{w}'$. Since f and g agree on all vectors from M, and $\mathbf{u}$ is stable, it also follows that f and g agree on some relevance hypercube for $\mathbf{u}$. This concludes the proof.

More than once we will need subsets of input variables having specific structure. We call a set $\mathbf{u}$ *conservative* iff for each internal node v of T labeled with a prime function h the number of subtrees of v containing at least one variable from $\mathbf{u}$ is equal either to 0, or to 1, or to the arity of h. To understand the intuition behind this term, consider the restriction of f to any relevance hypercube for such a set. In the tree of such a restriction, each node of T labeled with a prime function is either preserved or discarded, i. e., no constant substitutions and further transformations occur at these nodes.

Lemma 2. *All conservative sets are stable.*

Proof. Let $\mathbf{u}$ be a conservative set of variables. We say that an internal node of T is a *branching node* for $\mathbf{u}$ iff at least two subtrees of v contain leaves from $\mathbf{u}$. The proof is by induction over the number b of branching nodes for $\mathbf{u}$ in T. If $b = 0$, then $|\mathbf{u}| \leq 1$ and there is nothing to prove. Suppose that $b \geq 1$ and $v = \mathrm{lca}(\mathbf{u})$. Then v is a branching node and all other branching nodes are descendants of v. Therefore, f can be expressed by a formula

$$h_0\big(\mathbf{z}^0,\ h(h_1(\mathbf{z}^1), \ldots, h_m(\mathbf{z}^m))\big),$$

where h is the function corresponding to the label of v, sets $\mathbf{z}^i$ and $\mathbf{z}^j$ are disjoint for $i \neq j$ and (by our definition of a conservative set) variables $\mathbf{u}$ are contained in each $\mathbf{z}^i$, $i \geq 1$, but not in $\mathbf{z}^0$.

Identify a relevance hypercube H for some variables $\mathbf{w}$ (here $\mathbf{u} \subseteq \mathbf{w}$) with a partial assignment p of constants to variables $\mathbf{x}$. Split p into $p_0, p_1, \ldots, p_m$ according to the partitioning given by $\mathbf{z}^0, \mathbf{z}^1, \ldots, \mathbf{z}^m$. Subsets of $\mathbf{u}$ contained in $\mathbf{z}^i$, $i \geq 1$, are conservative for trees representing functions $h_i(\mathbf{z}^i)$, and the number of branching nodes in any such tree is at most $b - 1$. By the inductive assumption, there exist partial assignments $p'_1, \ldots, p'_m$ which are extensions of $p_1, \ldots, p_m$ and restrict relevance hypercubes for these subsets. Projections h'_i induced by p'_i depend on these subsets of $\mathbf{u}$. If we now choose an extension p'_0 of p_0 taking $h_0(\mathbf{z}^0, u)$ to a literal of u, the composition of $p'_0, p'_1, \ldots, p'_m$ will restrict the needed relevance hypercube H'. This concludes the proof.

In Section 5, we defined the colouring of variables induced by the read-once representation of g. The following lemmas reveal some properties of this colouring that are related to the structure of T. These properties reflect the observation formulated in Proposition 1.

Lemma 3. *Suppose that variables x and y both have colour k. Also suppose that the node $v = \mathrm{lca}(x, y)$ in T is labeled with a prime function h. Then all the leaves in the subtree T_v have the same colour k.*

Proof. Let m be the arity of h. Take arbitrary variables $z_1, \ldots, z_{m-2}$ such that $x, y, z_1, \ldots, z_{m-2}$ are contained in m different subtrees of v. The set $\mathbf{u} = \{x, y, z_1, \ldots, z_{m-2}\}$ is conservative and, therefore, stable (by Lemma 2). Since h is prime and f is read-once over B_l, we see that $m \leq l$. It follows from Lemma 1 that f agrees with g on some relevance hypercube for $\mathbf{u}$. By Proposition 3, the restriction of f to any such hypercube is similar to h. Therefore, some projection g' of g is represented by a tree T' with exactly one internal node, which is labeled with a prime function. Variables x and y are relevant to g' and have the same colour k. It then follows from Proposition 1 that all variables $z_1, \ldots, z_{m-2}$ have colour k too. Since $z_1, \ldots, z_{m-2}$ were chosen arbitrarily from their subtrees, we obtain that all leaves in $T_v^{z_1}, \ldots, T_v^{z_{m-2}}$ have colour k. Repeating the same reasoning for initial pairs x, z_1 and y, z_1 in place of x, y reveals that all leaves in T_v^y and T_v^x also have the same colour k. This concludes the proof.

Lemma 4. *For each colour $k \in \{1, \ldots, r\}$, there exists a unique index $i \in \{1, \ldots, s\}$ such that $\mathbf{y}^k \subseteq \mathbf{x}^i$, i. e., all variables coloured with k belong to the set $\mathbf{x}^i$.*

Proof. Take any two variables x and y having colour k. If they do not belong to the same $\mathbf{x}^i$, then they belong to different root subtrees of T. Therefore, the node $\mathrm{lca}(x, y)$ is labeled with a prime function f^0. By Lemma 3, all leaves of T have the same colour, which is a contradiction.

Another way to state Lemma 4 is to say that the partition $\mathbf{y}^1 \cup \ldots \cup \mathbf{y}^r$ is a refinement of $\mathbf{x}^1 \cup \ldots \cup \mathbf{x}^s$.

Lemma 5. *For any non-root internal node v in T labeled with a prime function h of arity r or greater, all leaves of T_v have the same colour.*

Proof. Let v be an internal node of T labeled with a prime function h of arity at least r. If not all leaves of T_v have the same colour, then by Lemma 3 any two leaves from different subtrees of v have different colours. It then follows that leaves of T_v are coloured with at least r colours. By Lemma 4, leaves of other root subtrees of T cannot be coloured, since all colours are taken from the set $\{1, \ldots, r\}$. This contradiction concludes the proof.

Lemma 6. *Suppose that variables x and y both have colour k. Also suppose that in T the node $v = \mathrm{lca}(x, y)$ is labeled with a symbol $\circ \in \{\wedge, \vee, \oplus\}$. Then all the leaves in T_v^x and T_v^y have the same colour k.*

Proof. Define the *depth* of a subtree as the maximum number of edges on (shortest) paths from its root to its leaves. Let d be the depth of T_v. The proof is by induction over d. For $d = 1$, there is nothing to prove. Suppose that $d \geq 2$ and T_v^y contains a leaf z having colour $k' \neq k$. We claim that for some $m \geq 0$ there exist variables $z_0, z_1, \ldots, z_m$ such that the set $\mathbf{u} = \{x, y, z_0, z_1, \ldots, z_m\}$ is conservative, has cardinality at most l, and the restriction of f to any relevance hypercube for $\mathbf{u}$ can be obtained by negating the variables and/or the output of some function $x \circ f'(y, z_0, z_1, \ldots, z_m)$, where the colours of y and z_0 are different and f' is either a prime function or a binary function ($m = 0$) from $\{\wedge, \vee, \oplus\}$ different from $\circ$.

First suppose that the root w of T_v^y is labeled with a prime function h. Since not all leaves of T_v^y have the same colour, it follows from Lemma 3 that colours of leaves taken from different subtrees of T_v^y are different. Take arbitrary variables $z_0, z_1, \ldots, z_m$, $m \geq 1$, from all subtrees except T_w^y (one variable from each subtree). Now y and z_0 have different colours and $m + 2 \leq r - 1$ by Lemma 5. One can easily see that the set $\mathbf{u}$ constructed in this way is conservative by definition and has cardinality at most l, because $r \leq l$. Proposition 3 then reveals that the restriction of f to any relevance hypercube for $\mathbf{u}$ indeed has the needed form.

Now consider the case when w is labeled with a symbol $\star \in \{\wedge, \vee, \oplus\}$. By definition of a tree representing a read-once function, $\star$ is different from $\circ$. Observe that the depth of the subtree T_w is less than or equal to $d - 1$, so we can use the inductive assumption for T_w. If y and z belong to the same subtree T' of w,

then it follows that only leaves from T' can have the same colour as y. In this case, any leaf from any other subtree can be chosen to be z_0. In the other case, if y and z belong to different subtrees, simply put $z_0 = z$. One can now see that $m = 0$ and $\mathbf{u} = \{x, y, z_0\}$ satisfy all the stated conditions.

Now apply Lemma 1 to the set $\mathbf{u}$ (recall that all conservative sets are stable by Lemma 2). It follows that g agrees with f on some relevance hypercube for $\mathbf{u}$. By our choice of $\mathbf{u}$, this means that g has a projection g' of the form specified above. In the tree of g', the root is adjacent to the leaf labeled with a literal of x and to the other internal node, whose children are $y, z_0, z_1, \ldots, z_m$. Since x and y have the same colour, it follows from Proposition 1 that z_0 has the same colour as x, which contradicts our choice of z_0. This completes the proof.

8 Main Theorem

Theorem. *For any read-once function f over B_l, $l \geq 3$, depending on all its variables, any relevance hypercube set of dimension l for f constitutes a checking test for f.*

Proof. Let M be a relevance hypercube set of dimension l for f. By g denote an alternative read-once function that agrees with f on all vectors from M. Suppose that f and g satisfy all the assumptions made in Section 5 and let T be a tree of f. Choose a subset $\mathbf{u}$ of T's leaves according to the following (non-deterministic) rules:

1. Put $\mathbf{u} = \mathbf{u}(v_0)$, where v_0 is the root of T.
2. If all leaves in T_v have the same colour, then $\mathbf{u}(v) = \{x_i\}$ for some leaf x_i contained in T_v.
3. Otherwise:
   (a) if v is labeled with a prime function h, then $\mathbf{u}(v) = \bigcup \mathbf{u}(v_i)$ over all children v_i of v;
   (b) otherwise, if v is labeled with a symbol $\circ \in \{\wedge, \vee, \oplus\}$, then $\mathbf{u}(v) = \mathbf{u}'(v) \cup \mathbf{u}''(v)$, where $\mathbf{u}'(v) = \bigcup \mathbf{u}(v_i)$ over all multi-coloured subtrees T_{v_i} of v, and $\mathbf{u}''(v) = \bigcup \mathbf{u}(v_j)$ over some subset of all single-coloured subtrees T_{v_j} of v that contains one subtree of each colour.

One can easily see that $\mathbf{u}$ is conservative and, by Lemma 2, stable. By Lemmas 3 and 6, it contains exactly r leaves. It follows from Lemma 1 that g agrees with f on some relevance hypercube H for $\mathbf{u}$. Since all elements of $\mathbf{u}$ have different colours, it follows from Proposition 3 that the restriction of g to H is similar to g^0. On the other hand, $\mathbf{u}$ contains at least one leaf from each root subtree of T, so the restriction of f to H has the form

$$f' = f^0(f'_1, \ldots, f'_s),$$

where functions f'_i depend on disjoint sets of variables from $\mathbf{u}$. These two restrictions are equal, so f' is a prime function, which is only possible if $r = s$ and

g^0 is similar to f^0. By Lemma 4, sets of root subtrees' variables are the same for f and g. This means that

$$g(\mathbf{x}) = f^0(g'_1(\mathbf{x}^1), \ldots, g'_s(\mathbf{x}^s)).$$

Since a relevance hypercube set for f contains relevance hypercube sets for all functions $f_1(\mathbf{x}^1), \ldots, f_s(\mathbf{x}^s)$ (or their negations) regarded as projections of f, the whole argument can be repeated recursively. In the end, one sees that f and g can be expressed by the same formula, and so $f = g$. This concludes the proof.

Corollary. *For any read-once function f over B_l depending on n variables it holds that*

$$T_{B_l}(f) \leq 2^l \cdot \binom{n}{l} = O(n^l).$$

9 Discussion

It is interesting to note that our result gives a stronger form of Kuznetsov's classic theorem on read-once representations [13]. The original result can be reformulated as follows: for any given Boolean function f and any two trees T_1 and T_2 representing f, there exists a one-to-one correspondence ϕ between the sets of internal nodes of T_1 and T_2 such that the functions represented by each pair of matching nodes are either equal or each other's negations. This fact was independently proved by Aaronson [1], who also developed an $O(N^{\log_2 3} \log N)$ algorithm for transforming the truth table of f into such a tree. Note that a sequence of $O(N)$-sized circuits that check the existence of and output read-once representations over B_l for any fixed l was constructed in [15]. In these results $N = 2^n$ is the input length.

Now suppose T_1 and T_2 are trees representing n-variable Boolean functions f_1 and f_2, and it is known a priori that these trees do not contain nodes labeled with prime functions of arity greater than l. Our technique reveals that in order to prove the existence of a correspondence ϕ it is sufficient to verify that f_1 and f_2 agree on an $O(n^l)$-sized set of input vectors. While Kuznetsov's theorem does not concern itself with computational issues, our theorem shows that only a small fraction of input vectors (in fact, a polynomial number of them, as compared to the total of 2^n) is needed to certify the "similarity" of the trees.

Acknowledgements. The author is indebted to Prof. Andrey A. Voronenko, who suggested the problem considered in this paper. The author also wishes to thank Maksim A. Bashov for useful discussions and the anonymous referees for their advice. This research has been supported by Russian Presidential grant MD–757.2011.9.

References

1. Aaronson, S.: Algorithms for Boolean function query properties. SIAM Journal on Computing 32(5), 1140–1157 (2003)
2. Angluin, D.: Queries and concept learning. Machine Learning 2(4), 319–342 (1988)
3. Angluin, D., Hellerstein, L., Karpinski, M.: Learning read-once formulas with queries. Journal of the ACM 40, 185–210 (1993)
4. Bshouty, N.H., Hancock, T.R., Hellerstein, L.: Learning Boolean read-once formulas over generalized bases. Journal of Computer and System Sciences 50(3), 521–542 (1995)
5. Bubnov, S.E., Voronenko, A.A., Chistikov, D.V.: Some test length bounds for non-repeating functions in the $\{\&, \vee\}$ basis. Computational Mathematics and Modeling 21(2), 196–205 (2010)
6. Chegis, I.A., Yablonsky, S.V.: Logical methods for controlling electrical circuits. Trudy Matematicheskogo Instituta Steklova 51, 270–360 (1958) (in Russian)
7. Chistikov, D.V.: On the relationship between diagnostic and checking tests of the read-once functions. Discrete Mathematics and Applications 21(2), 203–208 (2011)
8. Chistikov, D.V.: Read-once functions with hard-to-test projections. Moscow University Computational Mathematics and Cybernetics 34(4), 188–190 (2010)
9. Chistikov, D.V.: Testing Monotone Read-Once Functions. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2011. LNCS, vol. 7056, pp. 121–134. Springer, Heidelberg (2011)
10. Chistikov, D.V.: Testing read-once functions over the elementary basis. Moscow University Computational Mathematics and Cybernetics 35(4), 189–192 (2011)
11. Davies, R.O.: Two theorems on essential variables. Journal of the London Mathematical Society 41(2), 333–335 (1966)
12. Goldman, S.A., Kearns, M.J.: On the complexity of teaching. Journal of Computer and System Sciences 50(1), 20–31 (1995)
13. Kuznetsov, A.V.: On read-once switching circuits and read-once compositions of functions in the algebra of logic. Trudy Matematicheskogo Instituta Steklova 51, 186–225 (1958) (in Russian)
14. Voronenko, A.A.: On checking tests for read-once functions. In: Matematicheskie Voprosy Kibernetiki, vol. 11, pp. 163–176. Fizmatlit, Moscow (2002) (in Russian)
15. Voronenko, A.A.: Recognizing the nonrepeating property in an arbitrary basis. Computational Mathematics and Modeling 18(1), 55–65 (2007)
16. Voronenko, A.A.: Testing disjunction as a read-once function in an arbitrary unrepeated basis. Moscow University Computational Mathematics and Cybernetics 32(4), 239–240 (2008)
17. Voronenko, A.A., Chistikov, D.V.: Learning read-once functions individually. Uchenye Zapiski Kazanskogo Universiteta, ser. Fiziko-Matematicheskie Nauki 151(2), 36–44 (2009) (in Russian)
18. Voronenko, A.A., Chistikov, D.V.: On testing read-once Boolean functions in the basis B_5. In: Proceedings of the XVII International Workshop "Synthesis and Complexity of Control Systems", pp. 24–30. Izdatel'stvo Instituta matematiki, Novosibirsk (2008) (in Russian)
19. Zubkov, O.V., Chistikov, D.V., Voronenko, A.A.: An upper bound on checking test complexity for almost all cographs. In: Wang, D., et al. (eds.) 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2011), pp. 323–330. IEEE Computer Society, Los Alamitos (2012)

Approximating Minimum Power Edge-Multi-Covers

Nachshon Cohen and Zeev Nutov

The Open University of Israel
nachshonc@gmail.com, nutov@openu.ac.il

Abstract. Given a graph with edge costs, the *power* of a node is the maximum cost of an edge incident to it, and the power of a graph is the sum of the powers of its nodes. Motivated by applications in wireless networks, we consider the following fundamental problem in wireless network design. Given a graph $G = (V, E)$ with edge costs and degree bounds $\{r(v) : v \in V\}$, the Minimum-Power Edge-Multi-Cover (MPEMC) problem is to find a minimum-power subgraph J of G such that the degree of every node v in J is at least $r(v)$. Let $k = \max_{v \in V} r(v)$. For $k = \Omega(\log n)$, the previous best approximation ratio for MPEMC was $O(\log n)$, even for uniform costs [3]. Our main result improves this ratio to $O(\log k)$ for general costs, and to $O(1)$ for uniform costs. This also implies ratios $O(\log k)$ for the Minimum-Power k-Outconnected Subgraph and $O\left(\log k \log \frac{n}{n-k}\right)$ for the Minimum-Power k-Connected Subgraph problems; the latter is the currently best known ratio for the min-cost version of the problem. In addition, for small values of k, we improve the previously best ratio $k + 1$ to $k + 1/2$.

1 Introduction

1.1 Motivation and Problems Considered

Wireless networks are studied extensively due to their wide applications. The power consumption of a station determines its transmission range, and thus also the stations it can send messages to; the power typically increases at least quadratically in the transmission range. Assigning power levels to the stations (nodes) determines the resulting communication network. Conversely, given a communication network, the power required at v only depends on the farthest node reached directly by v. This is in contrast with wired networks, in which every pair of stations that communicate directly incurs a cost. An important network property is fault-tolerance, which is often measured by minimum degree or node-connectivity of the network. Node-connectivity is much more central here than edge-connectivity, as it models stations failures. Such power minimization problems were vastly studied; see for example [1,2,5,8,9] and the references therein for a small sample of papers in this area. The first problem we consider is finding a low power network with specified lower degree bounds. The second problem is the Min-Power k-Connected Subgraph problem. We give approximation algorithms for these problems, improving the previously best known ratios.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 64–75, 2012.

Definition 1. *Let (V, J) be a graph with edge-costs $\{c(e) : e \in J\}$. For a node $v \in V$ let $\delta_J(v)$ denote the set of edges incident to v in J. The* power *$p_J(v)$ of v is the maximum cost of an edge in J incident to v, or 0 if v is an isolated node of J; i.e., $p_J(v) = \max_{e \in \delta_J(v)} c(e)$ if $\delta_J(v) \neq \emptyset$, and $p_J(v) = 0$ otherwise. For $V' \subseteq V$ the power of V' w.r.t. J is the sum $p_J(V') = \sum_{v \in V'} p_J(v)$ of the powers of the nodes in V'.*

Unless stated otherwise, all graphs are assumed to be undirected and simple. Let $n = |V|$. Given a graph $G = (V, E)$ with edge-costs $\{c(e) : e \in E\}$, we seek to find a low power subgraph (V, J) of G that satisfies some prescribed property. One of the most fundamental problems in Combinatorial Optimization is finding a minimum-cost subgraph that obeys specified degree constraints (sometimes called also "matching problems") c.f. [10]. Another fundamental property is fault-tolerance (connectivity). In fact, these problems are related, and we use our algorithm for the former as a tool for approximating the latter.

Definition 2. *Given degree bounds $r = \{r(v) : v \in V\}$, we say that an edge-set J on V is an* r-edge cover *if $d_J(v) \geq r(v)$ for every $v \in V$, where $d_J(v) = |\delta_J(v)|$ is the degree of v in the graph (V, J).*

Minimum-Power Edge-Multi-Cover (MPEMC):
Instance: A graph $G = (V, E)$ with edge-costs $\{c(e) : e \in E\}$, degree bounds $r = \{r(v) : v \in V\}$.
Objective: Find a minimum power r-edge cover $J \subseteq E$.

Given an instance of MPEMC, let $k = \max_{v \in V} r(v)$ denote the maximum requirement.

We now define our connectivity problems. A graph is *k-outconnected from s* if it contains k internally-disjoint sv-paths for all $v \in V \setminus \{s\}$. A graph is *k-connected* if it is k-outconnected from every node, namely, if it contains k internally-disjoint uv-paths for all $u, v \in V$.

Minimum-Power k-Outonnected Subgraph (MPkOS):
Instance: A graph $G = (V, E)$ with edge-costs $\{c(e) : e \in E\}$, a root $s \in V$, and an integer k.
Objective: Find a minimum-power k-outconnected from s spanning subgraph J of G.

Minimum-Power k-Connected Subgraph (MPkCS):
Instance: A graph $G = (V, E)$ with edge-costs $\{c(e) : e \in E\}$ and an integer k.
Objective: Find a minimum-power k-connected spanning subgraph J of G.

1.2 Our Results

For large values of $k = \Omega(\log n)$, the previous best approximation ratio for MPEMC was $O(\log n)$, even for uniform costs [3]. Our main result improves this ratio to $O(\log k)$ for general costs, and to $O(1)$ for uniform costs.

Theorem 1. MPEMC *admits an* $O(\log k)$*-approximation algorithm. For uniform costs,* MPEMC *admits a randomized approximation algorithm with expected approximation ratio* $\rho < 2.16851$*, where* ρ *is the real root of the qubic equation* $e(\rho-1)^3 = 2\rho$.

For small values of k, the problem admits also the ratios $k+1$ for arbitrary k [2], while for $k=1$ the best known ratio is $k+1/2 = 3/2$ [4]. Our second result extends the latter ratio to arbitrary k.

Theorem 2. MPEMC *admits a* $(k+1/2)$*-approximation algorithm.*

For small values of k, say $k \leq 6$, the ratio $(k+1/2)$ is better than $O(\log k)$ because of the constant hidden in the $O(\cdot)$ term. And overall, our paper gives the currently best known ratios for all values $k \geq 2$.

In [5] it is proved that an α-approximation for MPEMC implies an $(\alpha+4)$-approximation for MPkOS. The previous best ratio for MPkOS was $O(\log n) + 4 = O(\log n)$ [5] for large values of $k = \Omega(\log n)$, and $k+1$ for small values of k [9]. From Theorem 1 we obtain the following.

Theorem 3. MPkOS *admits an* $O(\log k)$*-approximation algorithm.*

In [2] it is proved that an α-approximation for MPEMC and a β-approximation for Min-Cost k-Connected Subgraph implies a $(\alpha+2\beta)$-approximation for MPkCS. Thus the previous best ratio for MPkCS was $2\beta + O(\log n)$ [3], where β is the best ratio for MCkCS (for small values of k better ratios for MPkCS are given in [9]). The currently best known value of β is $O\left(\log k \log \frac{n}{n-k}\right)$ [7], which is $O(\log k)$, unless $k = n - o(n)$. From Theorem 1 we obtain the following.

Theorem 4. MPkCS *admits an* $O(\beta + \log k)$*-approximation algorithm, where* β *is the best ratio for* MCkCS*. In particular,* MPkCS *admits an* $O\left(\log k \log \frac{n}{n-k}\right)$*-approximation algorithm.*

1.3 Overview of the Techniques

Let the *trivial solution* for MPEMC be obtained by picking for every node $v \in V$ the cheapest $r(v)$ edges incident to v. It is known and easy to see that this produces an edge set of power at most $(k+1) \cdot \mathsf{opt}$, see [2].

Our $O(\log k)$-approximation algorithm uses the following idea. Extending and generalizing an idea from [3], we show how to find an edge set $I \subseteq E$ of power $O(\mathsf{opt})$ such that for the residual instance, the trivial solution value is reduced by a constant fraction. We repeatedly find and add such an edge set I to the constructed solution, while updating the degree bounds accordingly to $r(v) \leftarrow \max\{r(v) - d_I(v), 0\}$. After $O(\log k)$ steps, the trivial solution value is reduced to opt, and the total power of the edges we picked is $O(\log k) \cdot \mathsf{opt}$. At this point we add to the constructed solution the trivial solution of the residual problem, which at this point has value opt, obtaining an $O(\log k)$-approximate solution.

Our algorithm for uniform costs has two phases. In the first phase we compute an optimal solution x to a certain LP-relaxation for the problem and round it to 1 with probability $\min\{\rho \cdot x, 1\}$. In the second phase we add to the obtained partial solution the trivial solution to the residual problem.

Our $(k + 1/2)$-approximation algorithm uses a two-stage reduction. The first reduction reduces MPEMC to a constrained version of MPEMC with $k = 1$, where we also have lower bounds ℓ_v on the power of each node $v \in V$; these lower bounds are determined by the trivial solution to the problem. We will show that a ρ-approximation algorithm to this constrained version implies a $(k - 1 + \rho)$-approximation algorithm for MPEMC. The second reduction reduces the constrained version to the Minimum-Cost Edge Cover problem with a loss of 3/2 in the approximation ratio. As Minimum-Cost Edge Cover admits a polynomial time algorithm, we get a ratio $\rho = 3/2$ for the constrained problem, which in turn gives the ratio $k - 1 + \rho = k + 1/2$ for MPEMC.

2 Proof of Theorem 1

2.1 Reduction to Bipartite Graphs

Let Bipartite MPEMC be the restriction of MPEMC to instances for which the input graph $G = (V, E)$ is a bipartite graph with sides A, B, and with $r(a) = 0$ for every $a \in A$ (so, only the nodes in B may have positive degree bound).

As in [3], we can reduce MPEMC to Bipartite MPEMC, by taking two copies $A = \{a_v : v \in V\}$ and $B = \{b_v : v \in V\}$ of V, for every edge $e = uv \in E$ adding the two edges $a_u b_v$ and $a_v b_u$ of cost $c(e)$ each, and for every $v \in V$ setting $r(b_v) = r(v)$ and $r(a_v) = 0$. It is proved in [3] that this reduction invokes a factor of 2 in the approximation ratio, namely, that a ρ-approximation for bipartite MPEMC implies a 2ρ-approximation for general MPEMC.

In the case of uniform costs, we can save a factor of 2 using a different reduction.

Proposition 1. *Ratio ρ for* Bipartite MPEMC *with unit costs implies ratio ρ for* MPEMC *with uniform costs.*

Proof. Clearly, the case of uniform costs is equivalent to the case of unit costs. Now we show that for unit costs, MPEMC can be reduced to Bipartite MPEMC. Let $G = (V, E), r$ be an instance of MPEMC with unit costs. If there is an edge $e = uv \in E$ with $r(u), r(v) \geq 1$ or with $r(u) = r(v) = 0$, then we can obtain an equivalent instance by removing e from G, and in the case $r(u), r(v) \geq 1$ also decreasing each of $r(u), r(v)$ by 1. Hence we may assume that every $e \in E$ has one end in $A = \{a \in V : r(a) = 0\}$ and the other end in $B = \{b \in V : r(b) \geq 1\}$. The statement follows. □

2.2 An $O(\log k)$-Approximation Algorithm for General Costs

Let opt denote the optimal solution value of a problem instance at hand. For $v \in V$, let w_v be the cost of the $r(v)$-th least cost edge incident to v in E if

$r(v) \geq 1$, and $w_v = 0$ otherwise. Given a partial solution J to Bipartite MPEMC let $r_J(v) = \max\{r(v) - d_J(v), 0\}$ be the *residual bound* of v w.r.t. J. Let

$$R_J = \sum_{b \in B} w_b r_J(b) \ .$$

The main step in our algorithm is given in the following lemma, which will be proved later.

Lemma 1. *There exists a polynomial time algorithm that given an edge set $J \subseteq E$, an integer τ, and a parameter $\gamma > 1$, either correctly establishes that $\tau < \mathsf{opt}$, or returns an edge set $I \subseteq E \setminus J$ such that $p_I(V) \leq (1+\gamma)\tau$ and $R_{J \cup I} \leq \theta R_J$, where $\theta = 1 - \left(1 - \frac{1}{\gamma}\right)\left(1 - \frac{1}{e}\right)$.*

Lemma 2. *Let $J \subseteq E$ and let $F \subseteq E \setminus J$ be an edge set obtained by picking $r_J(b)$ least cost edges in $\delta_{E \setminus J}(b)$ for every $b \in B$. Then $J \cup F$ is an r-edge-cover and: $p_F(B) \leq \mathsf{opt}$, $p_F(A) \leq R_J \leq k \cdot \mathsf{opt}$.*

Proof. Since F is an r_J-edge-cover, $J \cup F$ is an r-edge-cover. By the definition of F, for any r-edge-cover I, $p_F(b) \leq w_b \leq p_I(b)$ for all $b \in B$. In particular, if I is an optimal r-edge-cover, then

$$p_F(B) \leq \sum_{b \in B} w_b \leq \sum_{b \in B} p_I(b) = p_I(B) \leq \mathsf{opt} \ .$$

Also,

$$R_J = \sum_{b \in B} w_b r_J(b) \leq k \cdot \sum_{b \in B} w_b \leq k \cdot \mathsf{opt} \ .$$

Finally, $p_F(A) \leq R_J$ since

$$p_F(A) = \sum_{a \in A} p_F(a) \leq \sum_{a \in A} \sum_{e \in \delta_F(a)} c(e) = \sum_{e \in F} c(e) \leq \sum_{b \in B} w_b r_J(b) = R_J \ .$$

This concludes the proof of the lemma. □

Theorem 1 is deduced from Lemmas 1 and 2 as follows. We set γ to be constant strictly greater than 1, say $\gamma = 2$. Then $\theta = 1 - \frac{1}{2}\left(1 - \frac{1}{e}\right)$. Using binary search, we find the least integer τ such that the following procedure computes an edge set J satisfying $R_J \leq \tau$.

Initialization: $J \leftarrow \emptyset$.
Loop: Repeat $\lceil \log_{1/\theta} k \rceil$ times:
Apply the algorithm from Lemma 2:
- If it establishes that $\tau < \mathsf{opt}$ then return "ERROR" and STOP.
- Else do $J \leftarrow J \cup I$.

After computing J as above, we compute an edge set $F \subseteq E \setminus J$ as in Lemma 2. The edge-set $J \cup F$ is a feasible solution, by Lemma 2. We claim that for any

$\tau \geq \mathsf{opt}$ the above procedure returns an edge set J satisfying $R_J \leq \tau$; thus binary search indeed applies. To see this, note that $R_\emptyset \leq k \cdot \mathsf{opt}$ and thus

$$R_J \leq R_\emptyset \cdot \theta^{\lceil \log_{1/\theta} k \rceil} \leq k \cdot \mathsf{opt} \cdot 1/k = \mathsf{opt} \leq \tau \ .$$

Consequently, the least integer τ for which the above procedure does not return "ERROR" satisfies $\tau \leq \mathsf{opt}$. Thus $p_J(V) \leq \lceil \log_{1/\theta} k \rceil \cdot (1+\gamma) \cdot \tau = O(\log k) \cdot \mathsf{opt}$. Also, by Lemma 2, $p_F(V) \leq \mathsf{opt} + R_J \leq 2\mathsf{opt}$. Consequently,

$$p_{J \cup F}(V) \leq p_J(V) + p_F(V) = O(\log k) \cdot \mathsf{opt} + 2\mathsf{opt} = O(\log k) \cdot \mathsf{opt} \ .$$

In the rest of this section we prove Lemma 1. It is sufficient to prove the statement in the lemma for the residual instance $((V, E \setminus J), r_J)$ with edge-costs restricted to $E \setminus J$; namely, we may assume that $J = \emptyset$. Let $R = R_\emptyset = \sum_{b \in B} w_b r(b)$.

Definition 3. *An edge $e \in E$ incident to a node $b \in B$ is* τ-cheap *if* $c(e) \leq \frac{\tau\gamma}{R} \cdot w_b r(b)$.

Lemma 3. *Let F be an r-edge-cover, let $\tau \geq p_F(B)$, and let*

$$I = \bigcup_{b \in B} \{e \in \delta_E(b) : c(e) \leq \frac{\tau\gamma}{R} \cdot w_b r(b)\}$$

be the set of τ-cheap edges in E. Then $R_{I \cap F} \leq R/\gamma$ and $p_I(B) \leq \gamma\tau$.

Proof. Let $D = \{b \in B : \delta_{F \setminus I}(b) \neq \emptyset\}$. Since for every $b \in D$ there is an edge $e \in F \setminus I$ incident to b with $c(e) > \frac{\tau\gamma}{R} \cdot w_b r(b)$, we have $p_{F \setminus I}(b) \geq \frac{\tau\gamma}{R} \cdot w_b r(b)$ for every $b \in D$. Thus

$$\tau \geq p_F(B) \geq p_{F \setminus I}(B) = \sum_{b \in D} p_{F \setminus I}(b) \geq \tau \cdot \frac{\gamma}{R} \sum_{b \in D} w_b r(b) \ .$$

This implies $\sum_{b \in D} w_b r(b) \leq R/\gamma$. Note that for every $b \in B \setminus D$, $\delta_F(b) \subseteq \delta_I(b)$ and hence $r_{I \cap F}(b) = r_F(b) = 0$. Thus we obtain:

$$R_{I \cap F} = \sum_{b \in B} w_b r_{I \cap F}(b) = \sum_{b \in D} w_b r_{I \cap F}(b) \leq \sum_{b \in D} w_b r(b) \leq R/\gamma \ .$$

To see that $p_I(B) \leq \gamma\tau$ note that

$$p_I(B) = \sum_{b \in B} p_I(b) \leq \frac{\tau\gamma}{R} \sum_{b \in B} w_b r(b) = \frac{\tau\gamma}{R} \cdot R = \tau\gamma \ .$$

This concludes the proof of the lemma. □

In [3] it is proved that the following problem, which is a particular case of submodular function minimization subject to matroid and knapsack constraint (see [6]) admits a $\left(1 - \frac{1}{e}\right)$-approximation algorithm.

Bipartite Power-Budgeted Maximum Edge-Multi-Coverage (BPBMEM):
Instance: A bipartite graph $G = (A \cup B, E)$ with edge-costs $\{c(e) : e \in E\}$ and node-weights $\{w_v : v \in B\}$, degree bounds $\{r(v) : v \in B\}$, and a budget τ.
Objective: Find $I \subseteq E$ with $p_I(A) \leq \tau$ that maximizes

$$\mathsf{val}(I) = \sum_{v \in B} w_v \cdot \min\{d_I(v), r(v)\} \ .$$

The following algorithm computes an edge set as in Lemma 1.

1. Among the τ-cheap edges, compute a $\left(1 - \frac{1}{e}\right)$-approximate solution I to BPBMEM.
2. If $R_I \leq \theta R$ then return I, where $\theta = 1 - \left(1 - \frac{1}{\gamma}\right)\left(1 - \frac{1}{e}\right)$;
   Else declare "$\tau < \mathsf{opt}$".

Clearly, $p_I(A) \leq \tau$. By Lemma 3, $p_I(B) \leq \gamma\tau$. Thus $p_I(V) \leq p_I(A) + p_I(B) \leq (1+\gamma)\tau$.

Now we show that if $\tau \geq \mathsf{opt}$ then $R_I \leq \theta R$. Let F be the set of cheap edges in some optimal solution. Then $p_F(A) \leq \mathsf{opt} \leq \tau$. By Lemma 3 $R_F \leq R/\gamma$, namely, F reduces R by at least $R\left(1 - \frac{1}{\gamma}\right)$. Hence our $\left(1 - \frac{1}{e}\right)$-approximate solution I to BPBMEM reduces R by at least $R\left(1 - \frac{1}{e}\right)\left(1 - \frac{1}{\gamma}\right)$. Consequently, we have $R_I \leq R - R\left(1 - \frac{1}{e}\right)\left(1 - \frac{1}{\gamma}\right) = \theta R$, as claimed.

The proof of Theorem 1 for the case of general costs is complete.

2.3 A Constant Ratio Approximation Algorithm for Unit Costs

Bipartite MPEMC with unit costs is closely related to the Set-Multicover problem, that can be casted as follows.

Set-Multicover
Instance: A bipartite graph $G = (A \cup B, E)$ and demands $\{r(b) : b \in B\}$.
Objective: Find a subgraph H of G with $\deg_H(b) \geq r(b)$ for every $b \in B$; minimize $|H \cap A|$.

In fact, it is easy to see that Bipartite MPEMC with unit costs is equivalent to the following modification of Set-Multicover, where instead of minimizing $|H \cap A|$ we seek to minimize $|H \cap A| + |B|$; namely, the problem we consider is as follows.

Set-Multicover+
Instance: A bipartite graph $G = (A \cup B, E)$ and demands $\{r(b) : b \in B\}$.
Objective: Find a subgraph H of G with $\deg_H(b) \geq r(b)$ for every $b \in B$; minimize $|H \cap A| + |B|$.

Clearly, ratio ρ for Set-Multicover implies ratio ρ for Set-Multicover+. As Set-Multicover admits ratio $H(|B|)$, so is Set-Multicover+. On the other hand, Set-Multicover+ is APX-hard even for instances with $\max_{a \in A} \deg_G(a) = 3$, by a

reduction from 3-Set-Cover. If $|A| = O(|B|)$ then the problem is clearly approximable within a constant; but we may have $|A| >> |B|$, if $k = \max_{b \in B} r(b)$ is large. We prove the following theorem that implies the second part of Theorem 1, and is also of independent interest.

Theorem 5. Set-Multicover+ *admits a randomized approximation algorithm with expected approximation ratio* ρ*, where* $\rho < 2.16851$ *is the real root of the qubic equation* $e(\rho - 1)^3 = 2\rho$.

Let $\Gamma(a)$ denote the set of neighbors of a in G. Consider the following LP-relaxation for both Set-Multicover and Set-Multicover+

$$\min \left\{ \sum_{a \in A} x_a : \sum_{a \in \Gamma(b)} x_a \geq r(b) \ \forall b \in B, 0 \leq x_a \leq 1 \ \forall a \in A \right\}. \tag{1}$$

The value of a solution x to LP (1) is $x(A) = \sum_{a \in A} x_a$ in the Set-Multicover case, and $x(A) + |B|$ in the Set-Multicover+ case. Given a partial cover $S \subseteq A$, the residual demand of $b \in B$ is $r_S(b) = \max\{r(b) - |\Gamma(b) \cap S|, 0\}$. Let $\rho > 1$ be a parameter eventually set to be as in Theorem 5. Let $\gamma = \gamma(\rho) = \frac{(\rho-1)^2}{2\rho}$. Note that $\gamma = 1$ if, and only if, $\rho = 2 + \sqrt{3}$, and that the value of ρ in Theorem 5 is less than $2 + \sqrt{3}$. Let x be a feasible solution to LP (1), and let $S \subseteq A$ be obtained by choosing every $a \in A$ with probability $\min\{\rho \cdot x_a, 1\}$.

Lemma 4. *If* $x_a < 1/\rho$ *for all* $a \in A$ *then* $\Pr[r_S(b) \geq 1] \leq e^{-\gamma \cdot r(b)}$ *for every* $b \in B$.

Proof. Let $C(b) = \Gamma(b) \cap S$ be a random variable that counts the number of times b is "covered" by S. Clearly, $r_S(b) \geq 1$ if, and only if, $C(b) < r(b)$. The expectation of $C(b)$ is $\mu_b = \mathbb{E}[C(b)] = \sum_{a \in \Gamma(b)} \rho \cdot x_a \geq \rho \cdot r(b)$. Since the nodes in $\Gamma(b)$ are chosen independently, $C(b)$ is a sum of independent Bernoulli random variables. The statement now follows by applying the Chernoff bound:

$$\begin{aligned} \Pr[C(b) < r(b)] &= \Pr\left[C(b) < \left(1 - \frac{\rho - 1}{\rho}\right) \cdot \rho \cdot r(b)\right] \leq \\ &\leq \Pr\left[C(b) < \left(1 - \frac{\rho - 1}{\rho}\right) \cdot \mu_b\right] \leq \\ &\leq e^{-\frac{1}{2}\left(\frac{\rho-1}{\rho}\right)^2 \mu_b} \leq e^{-\gamma \cdot r(b)} . \end{aligned}$$

□

Corollary 1. $\mathbb{E}[r_S(B)] \leq f(\rho)|B|$*, where* $f(\rho) = \frac{1}{e^\gamma}$ *if* $1 < \rho \leq 2 + \sqrt{3}$*. and* $f(\rho) = e^{-\gamma}$ *if* $\rho \geq 2 + \sqrt{3}$.

Proof. Let $S' = \{a : x_a \geq 1/\rho\}$, let $r'(b) = r_{S'}(b) = \max\{r(b) - |\Gamma(b) \cap S'|, 0\}$, and let x' be defined by $x'_a = 0$ if $a \in S'$ and $x'_a = x_a$ otherwise. Note that x' is a

feasible solution to LP (1) with the residual requirements r', and that $x'_a < 1/\rho$ for all $a \in A$. Thus by Lemma 4 we have

$$\mathbb{E}\left[r'(B)\right] = \sum_{b \in B} \mathbb{E}\left[r'(b)\right] \leq \sum_{b \in B} \Pr\left[r'(b) \geq 1\right] \cdot r'(b) \leq \sum_{b \in B} e^{-\gamma \cdot r'(b)} \cdot r'(b) \ .$$

Let $z = r'(b) \geq 1$ and $f(z) = e^{-\gamma z} \cdot z$. Then $f'(z) = e^{-\gamma z}(1 - \gamma z)$. Hence in the range $z \geq 1$, the function $f(z)$ has maximum value:

- $\frac{1}{e\gamma}$ if $\gamma \leq 1$ (namely, if $1 < \rho \leq 2 + \sqrt{3}$), attained at $z = 1/\gamma$.
- $e^{-\gamma}$ if $\gamma \geq 1$ (namely, if $\rho \geq 2 + \sqrt{3}$), attained at $z = 1$.

The statement follows. □

Now we finish the proof of Theorem 5. The algorithm is as follows. We compute an optimal solution x to LP (1), and then an edge set S as in Corollary 1. For every $b \in B$ let A_b be a set of $r_S(b)$ neighbors in $\Gamma(b) \setminus S$, and let $S' = \bigcup_{b \in B} A_b$. The solution returned is $S \cup S'$. Note that $\mathbb{E}[|S'|] \leq \mathbb{E}[r_S(B)]$. Thus by Corollary 1, the expected size of our solution is bounded by

$$\mathbb{E}(|S|) + \mathbb{E}(r_S(B)) + |B| \leq \rho x(A) + f(\rho)|B| + |B| \leq \max\{\rho, f(\rho)+1\}(x(A)+|B|) \ .$$

Consequently, as $x(A) + |B|$ is a lower bound on the optimal solution value, the approximation ratio is bounded by $\max\{\rho, f(\rho) + 1\}$. Solving the equation $\rho = f(\rho) + 1$ for $f(\rho) = \frac{1}{e\gamma} = \frac{2\rho}{e(\rho-1)^2}$ gives the result.

The proof of Theorem 5, and thus also of Theorem 1 for the case of uniform cost, is complete.

3 Proof of Theorem 2

We say that an edge set $F \subseteq E$ *covers* a node set $U \subseteq V$, or that F is a U*-cover*, if $\delta_F(v) \neq \emptyset$ for every $v \in U$. Consider the following auxiliary problem:

Restricted Minimum-Power Edge-Cover

Instance: A graph $G = (V, E)$ with edge-costs $\{c(e) : e \in E\}$, $U \subseteq V$, and degree bounds $\{\ell_v : v \in U\}$.

Objective: Find a power assignment $\{\pi(v) : v \in V\}$ that minimizes $\sum_{v \in V} \pi(v)$, such that $\pi(v) \geq \ell_v$ for all $v \in U$, and such that the edge set $F = \{e = uv \in E : \pi(u), \pi(v) \geq c(e)\}$ covers U.

Later, we will prove the following lemma.

Lemma 5. Restricted Minimum-Power Edge-Cover *admits a* $3/2$*-approximation algorithm.*

Theorem 2 is deduced from Lemma 5 and the following statement.

Lemma 6. *If* Restricted Minimum-Power Edge-Cover *admits a* ρ*-approximation algorithm, then* Minimum-Power Edge-Multi-Cover *admits a* $(k-1+\rho)$*-approximation algorithm.*

Proof. Consider the following algorithm.

1. Let $\pi(v)$ be the power assignment computed by the ρ-approximation algorithm for Restricted Minimum-Power Edge-Cover with $U = \{v \in V : r(v) \geq 1\}$ and bounds $\ell_v = w_v$ for all $v \in U$. Let $F = \{e = uv \in E : \pi(u), \pi(v) \geq c(e)\}$.
2. For every $v \in V$ let I_v be the edge-set obtained by picking the least cost $r_F(v)$ edges in $\delta_{E \setminus F}(v)$ and let $I = \cup_{v \in V} I_v$.

Clearly, $F \cup I$ is a feasible solution to Minimum-Power Edge-Multi-Cover. Let opt denote the optimal solution value for Minimum-Power Edge-Multi-Cover. In what follows note that $\pi(V) \leq \rho \cdot \mathsf{opt}$ and that $\sum_{v \in V} w_v \leq \mathsf{opt}$. We claim that

$$p_{I \cup F}(V) \leq \pi(V) + (k-1) \cdot \mathsf{opt} \ .$$

As $\pi(V) \leq \rho \cdot \mathsf{opt}$, this implies $p_{I \cup F}(V) \leq (\rho + k - 1) \cdot \mathsf{opt}$.

For $v \in V$ let Γ_v be the set of neighbors of v in the graph (V, I_v). The contribution of I_v to the total power is at most $p_{I_v}(\Gamma_v) + p_{I_v}(v)$. Note that $\pi(v) \geq p_{I_v}(v)$ and $\pi(v) \geq p_F(v)$ for every $v \in V$, hence $p_{F \cup I_v}(v) \leq \pi(v)$. This implies

$$p_{F \cup I}(V) \leq \sum_{v \in V} (\pi(v) + p_{I_v}(\Gamma_v)) = \pi(V) + \sum_{v \in V} p_{I_v}(\Gamma_v) \ .$$

Now observe that $|\Gamma_v| = |I_v| = r_F(v) \leq k - 1$ and that $p_{I_v}(u) \leq w_v$ for every $u \in \Gamma_v$. Thus

$$p_{I_v}(\Gamma_v) \leq (k-1) \cdot w_v \quad \forall v \in V \ .$$

Finally, using the fact that $\sum_{v \in V} w_v \leq \mathsf{opt}$, we obtain

$$p_{F \cup I}(V) \leq \pi(V) + \sum_{v \in V} p_{I_v}(\Gamma_v) \leq \pi(V) + (k-1) \sum_{v \in V} w_v \leq \pi(V) + (k-1) \cdot \mathsf{opt} \ .$$

This finishes the proof of the lemma. □

In the rest of this section we prove Lemma 5.

We reduce Restricted Minimum-Power Edge-Cover to the following problem that admits an exact polynomial time algorithm, c.f. [10].

Minimum-Cost Edge-Cover
Instance: A multi-graph (possibly with loops) $G = (U, E)$ with edge-costs.
Objective: Find a minimum cost edge-set $F \subseteq E$ that covers U.

Our reduction is not approximation ratio preserving, but incurs a loss of 3/2 in the approximation ratio. That is, given an instance (G, c, U, ℓ) of Restricted Minimum-Power Edge-Cover, we construct in polynomial time an instance (G', c') of Minimum-Cost Edge-Cover such that the following holds:

(i) For any U-cover I' in G' corresponds a feasible solution π to (G, c, U, ℓ) with $\pi(V) \leq c'(I')$.
(ii) $\mathsf{opt}' \leq 3\mathsf{opt}/2$, where opt is an optimal solution value to Restricted Minimum-Power Edge-Cover and opt' is the minimum cost of a U-cover in G'.

Hence if I' is an optimal (min-cost) solution to (G', c'), then $\pi(V) \leq c'(I') \leq 3\mathsf{opt}/2$. Clearly, we may set $\ell_v = 0$ for all $v \in V \setminus U$. For $I \subseteq E$ let

$$D(I) = \sum_{v \in V} \max\{p_I(v) - \ell_v, 0\} \ .$$

Here is the construction of the instance (G', c'), where $G' = (U, E')$ and E' consists of the following three types of edges, where for every edge $e' \in E'$ corresponds a set $I(e') \subseteq E$ of one edge or of two edges.

1. For every $v \in U$, E' has a loop-edge $e' = vv$ with $c'(vv) = \ell_v + D(\{vu\})$ where vu is is an arbitrary chosen minimum cost edge in $\delta_E(v)$.
   Here $I(e') = \{vu\}$.
2. For every $uv \in E$ such that $u, v \in U$, E' has an edge $e' = uv$ with $c'(uv) = \ell_u + \ell_v + D((\{uv\})$.
   Here $I(e') = \{uv\}$.
3. For every pair of edges $ux, xv \in E$ such that $c(ux) \geq c(xv)$, E' has an edge $e' = uv$ with $c'(uv) = \ell_v + \ell_u + D(\{ux, xv\})$.
   Here $I(e') = \{ux, xv\}$.

Lemma 7. *Let $I' \subseteq E'$ be a U-cover in G', let $I = \cup_{e \in I'} I(e)$, and let π be a power assignment defined on V by $\pi(v) = \max\{p_I(v), \ell_v\}$. Then $\pi(V) \leq c'(I')$, I is a U-cover in G, and π is a feasible solution to (G, c, U, ℓ).*

Proof. We have that I is a U-cover in G, by the definition of I and since $I(e')$ covers both endnodes of every $e' \in E'$. By the definition of π, we have that $I \subseteq \{e = uv \in E : \pi(u), \pi(v) \geq c(e)\}$. Hence π is a feasible solution to (G, c, U, ℓ), as claimed.

We prove that $\pi(V) \leq c'(I')$. For $e' = uv \in E'$ let $\ell(e') = \ell_v$ if e' is a type 1 edge, and $\ell(e') = \ell_u + \ell_v$ otherwise. Note that $\pi(v) = \max\{p_I(v), \ell(v)\} = \ell_v + \max\{p_I(v) - \ell(v), 0\}$, hence

$$\pi(V) = \sum_{v \in U} \ell_v + \sum_{v \in V} \max\{p_I(v) - \ell(v), 0\} = \sum_{v \in U} \ell_v + D(I) \ .$$

By the definition of $\ell(e')$ and since I' is a U-cover $\sum_{v \in U} \ell_v \leq \sum_{e' \in I'} \ell(e')$. Also, $D(I) = D\left(\cup_{e' \in I'} I(e')\right)$, by the definition of I. Thus we have

$$\sum_{v \in U} \ell_v + D(I) \leq \sum_{e' \in I'} \ell(e') + D\left(\cup_{e' \in I'} I(e')\right) \ .$$

It is easy to see that $D\left(\cup_{e' \in I'} I(e')\right) \leq \sum_{e' \in I'} D(I(e'))$. Finally, note that $\ell(e') + D(I(e')) = c'(e')$ for every $e' \in I'$ (if e' is a type 1 edge, this follows from our assumption that $\ell_v \geq \min\{c(e) : e \in \delta_E(v)\}$). Combining we get

$$\pi(V) = \sum_{v \in U} \ell_v + D(I) \leq \sum_{e' \in I'} \ell(e') + D\left(\cup_{e' \in I'} I(e')\right) \leq$$
$$\leq \sum_{e' \in I'} \ell(e') + \sum_{e' \in I'} D(I(e')) = \sum_{e' \in I'} \left(\ell(e') + D(I(e'))\right) = \sum_{e' \in I'} c'(e') = c'(I') \ .$$

This finishes the proof of the lemma. □

We prove the following statement in the full version.

Lemma 8. *Let* $\{\pi(v) : v \in V\}$ *be a feasible solution to an instance* (G, c, U, ℓ) *of* Restricted Minimum-Power Edge-Cover. *Then there exists a* U*-cover* I' *in* G' *such that* $c'(I') \leq 3\pi(V)/2$.

As was mentioned, Lemmas 7 and 8 imply Lemma 5. Lemmas 5 and 6 imply Theorem 2, hence the proof of Theorem 2 is now complete.

4 Conclusions and Open Problems

The main results of this paper are two new approximation algorithm for MPEMC: one with ratio $O(\log k)$ for general costs, and the other with constant ratio for uniform costs. This improves the ratio $O(\log(nk)) = O(\log n)$ of [3]. We also gave a $(k + 1/2)$-approximation algorithm, which is better than our $O(\log k)$-approximation algorithm for small values of k (roughly $k \leq 6$).

The main open problem is whether for general costs, the ratio $O(\log k)$ shown in this paper is tight, or the problem admits a constant ratio approximation algorithm.

References

1. Althaus, E., Calinescu, G., Mandoiu, I., Prasad, S., Tchervenski, N., Zelikovsky, A.: Power efficient range assignment for symmetric connectivity in static ad-hoc wireless networks. Wireless Networks 12(3), 287–299 (2006)
2. Hajiaghayi, M., Kortsarz, G., Mirrokni, V., Nutov, Z.: Power optimization for connectivity problems. Math. Programming 110(1), 195–208 (2007)
3. Kortsarz, G., Mirrokni, V., Nutov, Z., Tsanko, E.: Approximating minimum-power degree and connectivity problems. Algorithmica 58 (2010)
4. Kortsarz, G., Nutov, Z.: Approximating minimum-power edge-covers and $2, 3$-connectivity. Discrete Applied Mathematics 157, 1840–1847 (2009)
5. Lando, Y., Nutov, Z.: On minimum power connectivity problems. J. of Discrete Algorithms 8(2), 164–173 (2010)
6. Lee, J., Mirrokni, V., Nagarajan, V., Sviridenko, M.: Maximizing nonmonotone submodular functions under matroid or knapsack constraints. SIAM J. Discrete Mathematics 23(4), 2053–2078 (2010)
7. Nutov, Z.: An almost $O(\log k)$-approximation for k-connected subgraphs. In: SODA, pp. 912–921 (2009)
8. Nutov, Z.: Approximating minimum power covers of intersecting families and directed edge-connectivity problems. Theoretical Computer Science 411(26-28), 2502–2512 (2010)
9. Nutov, Z.: Approximating minimum power k-connectivity. Ad Hoc & Sensor Wireless Networks 9(1-2), 129–137 (2010)
10. Schrijver, A.: Combinatorial Optimization, Polyhedra and Efficiency. Springer, Heidelberg (2004)

A Lower Bound on Circuit Complexity of Vector Function in $U_2^\star$

Evgeny Demenkov

St. Petersburg State University, Russia

Abstract. In 1973, Lamagna and Savage proved the following result. If $f_j : \{0,1\}^n \to \{0,1\}$ for $1 \le j \le m$ depends on at least two variables and if for $i \ne j$, $f_i \ne f_j$ and $f_i \ne \bar{f}_j$, then for any binary basis Ω,

$$C_\Omega(f_1, \dots f_m) \ge \min_j C_\Omega(f_j) + m - 1,$$

where $C_\Omega(f)$ is the minimal size of a circuit computing f in the basis Ω.

The main purpose of this paper is to give a better lower bound for the following case. Let $f : \{0,1\}^n \to \{0,1\}$ and $f_i = f \oplus x_i$ for $1 \le i \le n$. Assume that f is not a constant after any three substitutions $x_i = c_i$ for different variables. Then

$$C_{U_2}(f_1, \dots f_n) \ge \min_{i \ne j, c_i, c_j} C_{U_2}(f \mid_{x_i = c_i, x_j = c_j}) + 2n - O(1),$$

where $U_2 = B_2 \setminus \{\oplus, \equiv\}$. This implies a $7n$ lower bound on the circuit complexity over U_2 of $f_1, \dots, f_n$ if f has circuit complexity at least $5n$.

1 Introduction

By B_n^m we denote the set of all Boolean functions $f: \{0,1\}^n \to \{0,1\}^m$. B_n^1 is denoted just by B_n. A circuit is a directed acyclic graph. In this graph there are nodes of two types. Nodes of in-degree 0 are marked by variables from $\{x_1, \dots, x_n\}$ and are called inputs. Nodes of in-degree 2 are marked by some functions $\divideontimes$ from B_2 and are called gates. If the first parent computes g_1 and the second parent computes g_2, we say that this node computes a function $f = g_1 \divideontimes g_2$.

There are also k specially marked output gates. Thus, such a circuit computes a function from B_n^k. The size of a circuit is its number of gates. For $\Omega \subseteq B_2$, by $C_\Omega(f)$ we denote the minimum size of a circuit computing f, where every gate is marked by a function from Ω.

In 1949 Shannon showed that almost all Boolean functions have circuits of size only $\Omega(2^n/n)$ [1] . But yet only a few proofs of linear lower bounds for explicit functions are known.

* The author is supported in part by RFBR (grant 11-01-12135) and Computer Science Club scholarship.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 76–80, 2012.

The best lower bound $3n - o(n)$ for the basis B_2 was proved by Blum in 1984 [2]. The same lower bound was proved recently by Demenkov and Kulikov [3]. The current record lower bound $5n - o(n)$ for the basis $U_2 = B_2 \setminus \{\oplus, \equiv\}$ was given in 2002 by Iwama and Morizumi [4] (for (n,k)-*Strongly-Two-Dependent Boolean function*).

The same sad situation is for multiple outputs. In 1973 Lamagna and Savage proved [5] if we have $f_1, \ldots, f_m$ then the lower bound for any basis Ω is still linear $\min\limits_i C_\Omega(f_i) + m - 1$.

In this paper we are interested in $\Omega = U_2$. A circuit of the smallest size in U_2 contains only binary functions and variables so every binary function f of two variables g_1, g_2 in U_2 can be presented as $f = (g_1 \oplus c_1) \wedge (g_2 \oplus c_2) \oplus d$, where c_1, c_2, d are some constants in $\{0,1\}$.

We prove a $\min\limits_{i \neq j, c_i, c_j} C_{U_2}(f\,|_{x_i=c_i, x_j=c_j}) + 2n - O(1)$ lower bound on the circuit complexity of n Boolean function f_i for $1 \leq i \leq n$, where $f_i = x_i \oplus f$.

For a (n,k)-*Strongly-Two-Dependent Boolean function* (and $k = o(n)$) we obtain a $7n - o(n)$ lower bound. The main idea is similar to Lamagna and Savage.

We consider the topological ordering of gates and show that there are at least n output gates and n their parents after a certain gate.

2 New Lower Bound

Let $f \in B_n$. By f_i we define a function that is equal $f \oplus x_i$. We define a $F \in B_n^n$ as $F = (f_1, \ldots, f_n)$. Note that F depends on all variables. By M_f we define $\min\limits_{i \neq j, c_i, c_j} C_{U_2}(f\,|_{x_i=c_i, x_j=c_j})$.

Theorem 1. *Let f be the function in B_n, $n \geq 4$. Assume that for any three pairwise different variables x_i, x_j, x_k and three constant $c_i, c_j, c_k \in \{0,1\}$ $f\,|_{x_i=c_i, x_j=c_j, x_k=c_k}$ is not constant. Then*

$$C_{U_2}(F) \geq M_f + 2n - 5$$

.

Let C be a circuit that has a minimal size in U_2 basis and that computes F. By D_C we denote the gates, whose number under some topological ordering of the gates is more than $M_f - 5$. Note that if $M_f > 5$ then the size of C is $M_f - 5 + |D_C|$. So, enough to show that $|D_C|$ contains at least $2n$ different gates for proof our theorem.

The circuit C has n output gates computing $f_1, \ldots, f_n$. Denote them by that, it is $F_1, \ldots, F_n$ (F_i computes f_i). Then all these gates belong to D_C.

Lemma 1. *All $F_i \in D_C$.*

Proof. It is clear that $|C_{U_2}(f) - C_{U_2}(f \oplus x_i)| \leq 3$, because we can compute one from another using only three gates. So, $C_{U_2}(f \oplus x_i) \geq C_{U_2}(f) - 3 > M_f - 4$. F_i in topological ordering cannot have a number less than $C_{U_2}(f_i) + n$. □

Every F_i can not compute a constant or a variable (because f depends on at least 4 variables), so they have 2 parents. We show that another output gate cannot be one of these parents.

Lemma 2. *No output gate is a parent for another output gate.*

Proof. Let F_j be a parent of F_i. Then for some function g and constants c, d:

$$f_i = g(f_j \oplus c) \oplus d \text{ or that same } f \oplus x_i \oplus d = (f \oplus x_j \oplus c)g.$$

Consider a substitution $x_i = d, x_j = c \oplus 1$. Let $f' = f\mid_{x_i=d,x_j=c\oplus 1}$ and $g' = g\mid_{x_i=d,x_j=c\oplus 1}$. Then

$$f' = g'(f' \oplus 1)$$

But in this case f' cannot take the value 1 (otherwise in the left side we have 1, in the right 0). So f' equals zero, a contradiction. □

Assume that some gate G computes g and has two children F_i and F_j. And $f_i = (g \oplus c_1)h_1 \oplus d_1$ and $f_j = (g \oplus c_2)h_2 \oplus d_2$. Then g feeds f_i and f_j entry with the same constants. It means:

Lemma 3. *Assume that $f_i = (g \oplus c_1)h_1 \oplus d_1$ and $f_j = (g \oplus c_2)h_2 \oplus d_2$. Then $c_1 = c_2$.*

Proof. Assume $c_1 = 1, c_2 = 0$. Consider the following product:

$$(f_i \oplus d_1)(f_j \oplus d_2) = (g \oplus 1)h_1 g h_2 = 0$$

If we take a substitution $x_i = d_1, x_j = d_2$, then $(f\mid_{x_i=d_1,x_j=d_2})^2 = 0$. A contradiction. □

Now we prove that every F_i has a parent in D_C.

Lemma 4. *For any $i \in [1..n]$ one of the parents of F_i belongs to D_C.*

Proof. If both parents of F_i do not belong to D_C then we can compute both of them using at most $M_f - 5$ gates. We compute f_i using one gate and we can compute f with another three gates. So $C_{U_2}(f) \leq M_f - 5 + 1 + 3 < M_f$. A contradiction. □

Now we prove that if one parent is parent for any another output then another parent belongs to D_C too. Moreover, the second parent has no other output child.

Lemma 5. *For any $i \in [1..n]$ if one of the parents F_i have another child say F_j $(j \neq i)$ then another parent of F_i belongs to D_C.*

Proof. Let $f_i = g_1 g_2 \oplus d$ and $f_j = (g_1 \oplus c_1)h \oplus e$. By Lemma 3, $c_1 = 0$. Consider the sum of f_i and f_j:

$$x_i \oplus x_j \oplus d \oplus e = f_i \oplus f_j \oplus d \oplus e = g_1(g_2 \oplus h).$$

After assigning $x_i = d, x_j = 1 \oplus e$ in the left side we have 1, so the expression in the right side equals 1 and then $g_1 \mid_{x_i=d,x_j=1\oplus e}$ equals 1 too. Then

$$f_i \mid_{x_i=d,x_j=1\oplus e} \oplus d = g_1 \mid_{x_i=d,x_j=1\oplus e} g_2 \mid_{x_i=d,x_j=1\oplus e} = g_2 \mid_{x_i=d,x_j=1\oplus e}$$

Then we have $C_{U_2}(f \mid_{x_i=d,x_j=1\oplus e}) = C_{U_2}(g_2 \mid_{x_i=d,x_j=1\oplus e}) \geq M_f$. So the gate, computing g_2 belongs to D_C. □

Finally in the last lemma we show that second parents have no output gate as a child.

Lemma 6. *Every F_i has a parent that is not a parent for any other F_j.*

Proof. Proof by contradiction. Assume F_i has parents G_1 and G_2, that compute g_1 and g_2 respectively. W.l.o.g. $f_i = g_1 g_2 \oplus d$. Let G_1 and G_2 have children F_j and F_k respectively($i \neq j, i \neq k$). Consider the following cases.

1. If $j = k$ then we can assume $f_j = (g_1 \oplus c_1)(g_2 \oplus c_2) \oplus e$. By Lemma 3 both c_1, c_2 equal zero, so f_i differs by adding constant from f_j, a contradiction.
2. If $j \neq k$. Assume that $f_j = (g_1 \oplus c_1)h_1 \oplus e_1$, $f_k = (g_2 \oplus c_2)h_2 \oplus e_2$. By Lemma 3 $c_1 = 0, c_2 = 0$.

   $$(f_j \oplus e_1)(f_k \oplus e_2) = g_1 h_1 g_2 h_2 = (f_i \oplus d)h_1 h_2.$$

   Consider assigning $x_j = e_1, x_k = e_2, x_i = 1 \oplus d$. By f' we denote $f \mid_{x_j=e_1,x_k=e_2,x_i=1\oplus d}$ get

   $$f' = (f' \oplus 1)(h_1 h_2) \mid_{x_j=e_1,x_k=e_2,x_i=1\oplus d}.$$

   If for some assignment of variables f' equals 1, we have in the left side 1 and 0 in the right. So f' equals 0. A contradiction.

□

Finally we can prove the theorem.

Proof (of Theorem). It is enough to show that $|D_C| \geq 2n$ to prove the theorem. By Lemma 1, we have n output gates. By Lemma 4, every output gate has a parent in D_C. If this parent has another output gate as child then, by Lemma 6, F_i has another unique parent in D_C. So every F_i has unique parent in D_C. By Lemma 2, output gates and their unique parents are different $2n$ gates. Thus, the total number of gates is at least $M_f - 5 + |D_C| \geq M_f - 5 + 2n$. □

Acknowledgements. The author is grateful to his supervisor Alexander S. Kulikov for stating the problem and to Arist Kojevnikov for helpful discussion. Also to Olga Melanich, Ekaterina Lisnyak, Darya Lavrentyava, and anonymous referees for help in improving the readability of the paper.

References

1. Shannon, C.E.: The synthesis of two-terminal switching circuits. Bell System Technical Journal 28, 59–98 (1949)
2. Blum, N.: A Boolean function requiring $3n$ network size. Theoretical Computer Science 28, 337–345 (1984)
3. Demenkov, E., Kulikov, A.S.: An Elementary Proof of a $3n - o(n)$ Lower Bound on the Circuit Complexity of Affine Dispersers. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 256–265. Springer, Heidelberg (2011)
4. Iwama, K., Morizumi, H.: An Explicit Lower Bound of $5n - o(n)$ for Boolean Circuits. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 353–364. Springer, Heidelberg (2002)
5. Lamagna, E., Savage, J.: On the logical complexity of symmetric switching functions in monotone and complete bases. Discrete Optimization, Technical report, Brown University, Rhode Island (1973)

Computing All MOD-Functions Simultaneously*

Evgeny Demenkov[1], Alexander S. Kulikov[2,3], Ivan Mihajlin[4], and Hiroki Morizumi[5]

[1] St. Petersburg State University, Russia
[2] St. Petersburg Department of Steklov Institute of Mathematics, Russia
[3] Algorithmic Biology Laboratory, St. Petersburg Academic University
[4] St. Petersburg State Polytechnical University, Russia
[5] Shimane University, Japan

Abstract. The fundamental symmetric functions are EX_k^n (equal to 1 if the sum of n input bits is exactly k), TH_k^n (the sum is at least k), and $\mathrm{MOD}_{m,r}^n$ (the sum is congruent to r modulo m). It is well known that all these functions and in fact any symmetric Boolean function have linear circuit size.

Simple counting shows that the circuit complexity of computing n symmetric functions is $\Omega(n^{2-o(1)})$ for almost all tuples of n symmetric functions. It is well-known that all EX and TH functions (i.e., for all $0 \le k \le n$) of n input bits can be computed by linear size circuits.

In this short note, we investigate the circuit complexity of computing all MOD functions (for all $1 \le m \le n$). We prove an $O(n)$ upper bound for computing the set of functions

$$\{\mathrm{MOD}_{1,r}^n, \mathrm{MOD}_{2,r}^n, \ldots, \mathrm{MOD}_{n,r}^n\}$$

and an $O(n \log\log n)$ upper bound for

$$\{\mathrm{MOD}_{1,r_1}^n, \mathrm{MOD}_{2,r_2}^n, \ldots, \mathrm{MOD}_{n,r_n}^n\},$$

where $r_1, r_2, \ldots, r_n$ are arbitrary integers.

1 Introduction

By $B_{n,m}$ we denote the set of all Boolean functions $f\colon \{0,1\}^n \to \{0,1\}^m$. $B_{n,1}$ is denoted just by B_n. A function $f \in B_n$ is called *symmetric* if its value depends only on the sum of the input bits. That is, there must exist a vector $v \in \{0,1\}^{n+1}$ (called the *value vector* of f) such that $f(x_1, \ldots, x_n) = v_s$ where $s = \sum_{1 \le i \le n} x_i$.

* Research is partially supported by Russian Foundation for Basic Research (11-01-00760-a and 11-01-12135-ofi-m-2011), RAS Program for Fundamental Research, and JSPS KAKENHI 23700020. Part of this research was done while the second author was visiting the University of Kyoto.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 81–88, 2012.

The fundamental symmetric functions are EX, TH, and MOD defined as follows:

$$\mathrm{EX}^n_k(x_1,\ldots,x_n)=1 \text{ iff } \sum_{1\le i\le n} x_i = k\,,$$
$$\mathrm{TH}^n_k(x_1,\ldots,x_n)=1 \text{ iff } \sum_{1\le i\le n} x_i \ge k\,,$$
$$\mathrm{MOD}^n_{m,r}(x_1,\ldots,x_n)=1 \text{ iff } \sum_{1\le i\le n} x_i \equiv r \pmod m\,.$$

A circuit is a directed acyclic graph with nodes of in-degree 0 or 2. Nodes of in-degree 0 are marked by variables from $\{x_1,\ldots,x_n\}$ and are called inputs. Nodes of in-degree 2 are marked by functions from B_2 and are called gates. There are also m specially marked output gates. Thus, such a circuit computes a function from $B_{n,m}$. An example of a circuit computing a function from $B_{3,2}$ is given in Fig. 1 (page 84). The size of a circuit is its number of gates. By $C(f)$ we denote the minimum size of a circuit computing f.

It is known that the circuit complexity of almost all functions from B_n is $\Theta(2^n/n)$ [1,2]. The best lower bound proved for an *explicit* function is $3n-o(n)$ [3,4], i.e., just linear. The strongest lower bound for a symmetric function is $2.5n-O(1)$ [5]. Any symmetric function of n variables can be computed by linear size circuits (Corollary 1). The best known explicit upper bound is $4.5n+o(n)$ [6].

Note that currently we do not know any non-linear lower bounds on the circuit size even for functions from $B_{n,n}$. Simple counting shows that the circuit complexity of computing n symmetric functions is $\Omega(n^{2-o(1)})$ for almost all tuples of n symmetric functions. It is well-known (Corollary 1) that the sets of functions

$$\{\mathrm{EX}^n_0,\mathrm{EX}^n_1,\ldots,\mathrm{EX}^n_n\} \text{ and } \{\mathrm{TH}^n_0,\mathrm{TH}^n_1,\ldots,\mathrm{TH}^n_n\}\,.$$

can be computed by linear size circuits.

In this short note, we show that all MOD-functions for $1\le m\le n$ can also be computed simultaneously by circuits of small size. For n integers $r_1,r_2,\ldots,r_n$, by $\mathrm{ALLMOD}^n_{r_1,\ldots,r_n}$ we denote the set of functions

$$\{\mathrm{MOD}^n_{1,r_1},\mathrm{MOD}^n_{2,r_2},\ldots,\mathrm{MOD}^n_{n,r_n}\}\,.$$

If $r_1=r_2=\cdots=r_n=r$ we write just ALLMOD^n_r. We prove the following upper bounds:

$$C(\mathrm{ALLMOD}^n_{r_1,\ldots,r_n})=O(n\log\log n)\,,$$
$$C(\mathrm{ALLMOD}^n_r)=O(n)\,.$$

2 Preparations

In this section, we give some basic facts needed in our constructions. In most cases, we ignore integral parts in the estimates. This does not influence the

asymptotic behaviour of functions under consideration. By X we denote $\sum_{i=1}^{n} x_i$. We also usually omit n where it is clear from the context. By $\log x$ and $\ln x$ we denote logarithm x base 2 and e, respectively.

2.1 Basic Facts from Number Theory

Lemma 1. *1. Harmonic series ([7], Theorem 2.5.3):*

$$H_n = \sum_{1 \le k \le n} \frac{1}{k} = \ln n + \Theta(1)\,.$$

2. Harmonic series of primes ([7], Theorem 8.8.5):

$$P_n = \sum_{p \le n, p \in \mathbb{P}} \frac{1}{p} = \ln \ln n + \Theta(1).$$

3. The asymptotic law of distribution of prime numbers ([7], Theorem 8.8.1): the number of primes less or equal than n is $\pi(n) = \Theta(\frac{n}{\ln n})$.

2.2 Encodings

Given a number $m \in \mathbb{N}$, there are two natural ways to encode a residue number r modulo m by binary strings.

- $bin(r, m) = (b_0, b_1, \ldots, b_{\log m})$ is a $\log m$-bit binary representation of r:

$$\sum_{0 \le i \le \log m} b_i 2^i = r\,.$$

- $ex(r, m) = (e_0, e_1, \ldots, e_{m-1})$ is an m-bit string such that $e_r = 1$ and $e_k = 0$, for all $0 \le k \le m-1$, $k \ne r$.

Clearly, the former representation is more compact than the latter. At the same time it is easier to check whether a given residue number is equal to some particular number if it is given in the second representation. E.g., to check whether a string $b \in \{0,1\}^4$ represents 6 mod 10 one needs to compute the bit $(\neg b_0) \wedge b_1 \wedge b_2 \wedge (\neg b_3)$, while if it is given as a 10-bit string e one just needs to check the bit e_6.

2.3 Building Blocks

The following two lemmas are well-known and can be found, e.g., in [8] (Subsec. 3.4, pp. 85–87). Both of them are quite simple, so we sketch their proofs for completeness.

Lemma 2. *There exists a circuit of size $O(n)$ that takes n input bits $x_1, \ldots, x_n$ and outputs $bin(X, n+1)$, i.e., the binary representation of X.*

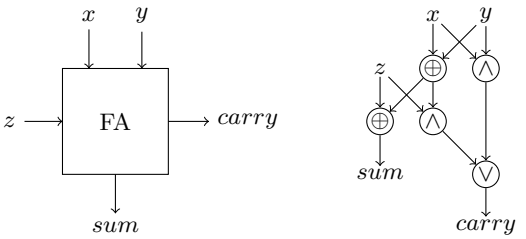


Fig. 1. Full Adder block

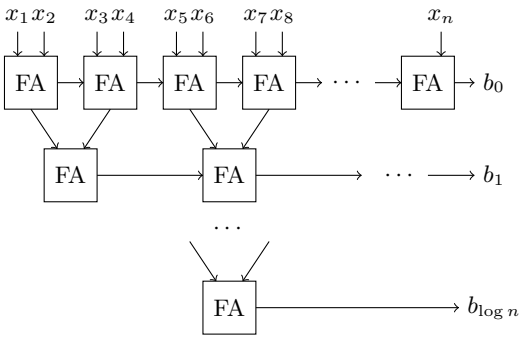


Fig. 2. The well-known way of computing the binary representation of the sum of n bits using $5n$ gates

Proof. The needed circuit can be built of the so-called Full Adders (FA). FA is a function from $B_{3,2}$ that for three input bits (x, y, z) outputs two bits $(carry, sum)$ such that $x + y + z = 2 \cdot carry + sum$. FA can be implemented using five gates (Fig. 1). One then needs $\log n$ layers of FA blocks to compute the binary representation of X (Fig. 2).

□

Lemma 3. *There exists a circuit of size* $O(m)$ *that for* $\log m$ *input bits* $(b_0, \ldots, b_{\log m}) = bin(r, m)$ *outputs* $(e_0, \ldots, e_m) = ex(r, m + 1)$.

Proof. Note that each e_j is just a conjunction $d_0 \wedge \cdots \wedge d_{\log m}$ where each d_j is either b_j or $\neg b_j$. Thus, we need to compute all these $m+1$ conjunctions in $O(m)$ gates. A straightforward computation results in a circuit of size $O(m \log m)$, since each conjunction requires $\log m$ binary AND-gates. To reduce the size to linear we first compute the set C_1 of all possible conjunctions of the first half of b (i.e., on bits $b_0, \ldots, b_{(\log m)/2}$) and the set C_2 of all possible conjunctions of the second half. This requires $O(\sqrt{m} \log m)$ gates. Then, any conjunction on all the bits of b is just an AND of two conjunctions from C_1 and C_2. □

Combining the previous two lemmas we get the following corollary.

Corollary 1. *There exists a circuit of size $O(n)$ that for n input bits $x_1, x_2, \ldots, x_n$ outputs $(e_0, \ldots, e_n) = ex(X, n+1)$.*

Corollary 1 immediately implies that any symmetric function f can be computed by linear size circuits. Indeed, let $(e_0, \ldots, e_n) = ex(X, n+1)$ and let $(v_0, \ldots, v_n)$ be the value vector of f. Then clearly

$$f(x_1, \ldots, x_n) = \bigvee_{0 \le i \le n} (e_i \wedge v_i) .$$

Another thing that follows from the corollary is that $\{\mathrm{EX}^n_k\}_{0 \le k \le n}$ and $\{\mathrm{TH}^n_k\}_{0 \le k \le n}$ can be computed by linear size circuits: EX^n_k is just e_k and

$$\mathrm{TH}^n_{k-1} = \mathrm{TH}^n_k \vee \mathrm{EX}^n_{k-1} .$$

Lemma 4. *For any m, $bin(X \bmod m, m)$ can be computed from $bin(X, n+1)$ in $O(\log n \log m)$ gates.*

Proof. We are given the binary representation $b = (b_0, \ldots, b_{\log n})$ of the sum of n input bits and we need to compute the binary representation of its residue number modulo m. Let $l = \log n$. Then $X = \sum_{o \le i \le l} b_i 2^i$.

We first build $(l+1)$ blocks $B_0, \ldots, B_l$. The block B_i takes the bit b_i as an input and outputs $\log m$ bits encoding the residue number modulo m of $b_i 2^i$ (i.e., $bin(b_i 2^i \bmod m, m)$). Note that this is a very simple block: its output is either 0 or $(2^i \bmod m)$. Clearly, $O(\log m)$ gates are enough to build it.

After that, we have to sum up our $(l+1)$ residue numbers. For this, we need another block A which takes two residue numbers a and b modulo m (i.e., $2 \log m$ input bits) and outputs $(a+b) \bmod m$ ($\log m$ bits). The block A also needs only $O(\log m)$ gates as we need to just compute the sum $(a+b)$ and then subtract m in case $a+b \ge m$. To compute the sum modulo m of $(l+1)$ residue numbers, we need l copies of the block A. The size of the resulting circuit is $O(\log m \log n)$. □

Lemma 5. *$MOD^n_{m,r}$ can be computed from $ex(X, n+1)$ in n/m gates.*

Proof. This is straightforward as

$$\mathrm{MOD}^n_{m,r} = \bigvee_{m \mid (q-r)} e_q .$$

□

Lemma 6. *$MOD^n_{m,r}$-functions for all $1 \le m \le \sqrt{n}$ and all $0 \le r < m$ can be computed in linear number of gates, i.e.,*

$$C\left(\left\{MOD^n_{m,r}\right\}_{0 \le r < m \le \sqrt{n}}\right) = O(n) .$$

Proof. By Lemma 4, $bin(X \bmod m, m)$ can be computed in $c \log n \log m$ gates. Hence, computing $bin(X \bmod m, m)$ for all $m \le \sqrt{n}$ requires

$$\sum_{1 \le m \le \sqrt{n}} c \log n \log m \le \sqrt{n} \cdot c \log n \log \sqrt{n} = o(n)$$

gates. We then compute

$$ex(X \bmod m, m) = \left\{\mathrm{MOD}^n_{m,r}\right\}_{0 \le r < m}$$

from $bin(X \bmod m, m)$ using $O(m)$ gates by Lemma 3. Summing over all $m \le \sqrt{n}$ gives a linear upper bound:

$$\sum_{1 \le m \le \sqrt{n}} O(m) = O(n)\,.$$

□

3 Upper Bounds

In this section, we describe small circuits for ALLMOD. We first present a relatively simple proof of an upper bound $C(\mathrm{ALLMOD}^n_r) = O(n)$ and then prove that $C(\mathrm{ALLMOD}^n_{r_1,\ldots,r_n}) = O(n \log\log n)$

Throughout the section, we assume that $bin(X, n+1), ex(X, n+1)$ and $\left\{\mathrm{MOD}^n_{m,r}\right\}_{0 \le r < m \le \sqrt{n}}$ are already computed (by Lemmas 2, 3, and 6 this requires $O(n)$ gates only).

3.1 $C(\mathrm{ALLMOD}^n_{\boldsymbol{r}}) = O(n)$

Below we prove a linear upper bound for the case when all r_i are equal. Since r_i's are equal, the problem boils down to computing $\mathrm{MOD}^n_{m,r}$ for all prime powers m. We compute MOD^n_{m,r_m} for prime powers $m = p^k$ in two steps: for $1 < p \le \sqrt{n}$ and $\sqrt{n} < p \le n$

Theorem 1. $C(\mathrm{ALLMOD}^n_r) = O(n)\,.$

Proof. We compute $\mathrm{MOD}^n_{m,r}$ for all $m = p^k$, where $p \le \sqrt{n}$ and $p \in \mathbb{P}$. There are at most $\sqrt{n} \log n$ such m's (since $k \le \log n$), hence by Lemma 4, the complexity of this is at most $O(\sqrt{n} \log^3 n)$.

We then compute $\mathrm{MOD}^n_{m,r}$ for all the remaining primes m (note that there are no prime powers $1 \le m = p^k \le n$ such that $\sqrt{n} < p \le n$ and $k > 1$). By Lemma 5, the complexity of this step is

$$\begin{aligned}\sum_{\sqrt{n} < p \le n,\, p \in \mathbb{P}} \frac{n}{p} &= \\ &= n(P_n - P_{\sqrt{n}}) = && \text{(Lemma 1.2)} \\ &= n(\ln\ln n - \ln\ln\sqrt{n} + \Theta(1)) = O(n)\,.\end{aligned}$$

We have computed $\mathrm{MOD}^n_{m,r}$ for all prime powers m. Then, for m from 2 to n, if m is not a prime power, represent it as a product st of two relatively prime numbers s and t. Then clearly

$$\mathrm{MOD}^n_{m,r} = \mathrm{MOD}^n_{s,r} \wedge \mathrm{MOD}^n_{t,r}\,.$$

Thus, each m that is not a prime power requires only one additional conjunction gate, so the size of the constructed circuit is $O(n)$. □

3.2 $C(\mathrm{ALLMOD}^n_{r_1,\ldots,r_n}) = O(n \log\log n)$

In this subsection, we prove an $O(n \log\log n)$ upper bound for $C(\mathrm{ALLMOD}^n_{r_1,\ldots,r_n})$. Again, we first compute $\mathrm{MOD}^n_{m,r}$ for prime powers m, but now only for $m \le n/\ln n$.

Theorem 2. $C(\mathrm{ALLMOD}^n_{r_1,\ldots,r_n}) = O(n \log\log n)$.

Proof. By Lemma 4, $c \log m \log q$ gates are enough to compute $bin(X \bmod q, q)$. We first compute the remainders of X modulo all primes $p \le n/\ln n$. The number of gates needed is at most (up to a constant factor)

$$\begin{aligned}
\sum_{2\le p\le n/\ln n, p\in\mathbb{P}} \log n \log p = \\
&= \log n \sum_{2\le p\le n/\ln n, p\in\mathbb{P}} \log p = \qquad \text{(Lemma 1.3)}\\
&= \log n \cdot O\left(\frac{n/\ln n}{\ln(n/\ln n)}\right) \cdot \log(n/\ln n) = \\
&= O(n)\,.
\end{aligned}$$

Now we compute the remainders of X modulo prime powers $p^k \le n/\ln n$ for $k \ge 2$. Since the number of such prime powers is at most $\sqrt{n}\log n$ this step takes at most

$$(\sqrt{n}\log n)\log n \log n = o(n)\,.$$

As a next step, we compute MOD^n_{m,r_m} for all $m \le n/\ln n$ that are not prime powers. Represent each such m as a product of prime powers: $m = q_1 \ldots q_k$. Then

$$\mathrm{MOD}^n_{m,r_m} = \bigwedge_{1\le i\le k} \mathrm{MOD}^n_{q_i,r_m}\,.$$

Each $\mathrm{MOD}^n_{q_i,r_m}$ can be computed from $bin(X \bmod q_i, q_i)$ in $O(\log q_i)$. Hence, to compute MOD^n_{m,r_m} one needs

$$\sum_{1\le i\le k} \log q_i = \log \prod_{1\le i\le k} q_i = \log m$$

gates. Overall, MOD^n_{m,r_m} for all $m \le n/\ln n$ can be computed in $O(n)$ gates.

Finally, we compute MOD^n_{m,r_m} for all the remaining m's, i.e., for $m \geq n/\ln n$. By Lemma 5, the number of gates needed for this step is at most

$$\sum_{n/\ln n \leq m \leq n} \frac{n}{m} =$$

$$= n(H_n - H_{n/\ln n}) = \qquad \text{(Lemma 1.1)}$$
$$= n(\ln n - \ln(n/\ln n) + \Theta(1)) =$$
$$= O(n \log\log n)\,.$$

□

4 Further Directions

1. The most natural question that is left open is whether the circuit complexity of $\mathrm{ALLMOD}^n_{r_1,\ldots,r_n}$ is linear or not. By using the same ideas and a more accurate analysis we were able to prove a stronger $O(n \log\log\log n)$. It seems that new ideas are needed for proving a linear upper bound.
2. The exact circuit complexity of MOD^n_m is known for $m = 2$ and $m = 4$ only [5]:
$$C(\mathrm{MOD}^n_2) = n - 1,\ C(\mathrm{MOD}^n_4) = 2.5n + \Theta(1)\,.$$
Also, it is known [5,6] that
$$2.5n - O(1) \leq C(\mathrm{MOD}^n_3) \leq 3n + O(1)\,.$$
For all the remaining m, we know a lower bound $2.5n - O(1)$ [5] and an upper bound $4.5n + o(n)$ [6]. It would be interesting to close these gaps.

References

1. Shannon, C.E.: The synthesis of two-terminal switching circuits. Bell System Technical Journal 28, 59–98 (1949)
2. Lupanov, O.: A method of circuit synthesis. Izvestiya VUZov, Radiofizika 1, 120–140 (1959) (in Russian)
3. Blum, N.: A Boolean function requiring $3n$ network size. Theoretical Computer Science 28, 337–345 (1984)
4. Demenkov, E., Kulikov, A.S.: An Elementary Proof of a $3n - o(n)$ Lower Bound on the Circuit Complexity of Affine Dispersers. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 256–265. Springer, Heidelberg (2011)
5. Stockmeyer, L.J.: On the combinational complexity of certain symmetric Boolean functions. Mathematical Systems Theory 10, 323–336 (1977)
6. Demenkov, E., Kojevnikov, A., Kulikov, A.S., Yaroslavtsev, G.: New upper bounds on the Boolean circuit complexity of symmetric functions. Information Processing Letters 110(7), 264–267 (2010)
7. Bach, E., Shallit, J.O.: Algorithmic Number Theory. Efficient Algroithms of Foundations of Computing, vol. I. MIT Press, Massachusetts (1996)
8. Wegener, I.: The Complexity of Boolean Functions. Wiley-Teubner (1987)

Bounded Synchronization Delay in Omega-Rational Expressions

Volker Diekert and Manfred Kufleitner*

FMI, University of Stuttgart, Germany
{diekert, kufleitner}@fmi.uni-stuttgart.de

Abstract. In 1965 Schützenberger published his famous result that star-free languages (SF) and aperiodic languages (AP) coincide over finite words, often written as SF = AP. Perrin generalized SF = AP to infinite words in the mid 1980s. In 1973 Schützenberger presented another (and less known) characterization of aperiodic languages in terms of rational expressions where the use of the star operation is restricted to prefix codes with bounded synchronization delay and no complementation is used. We denote this class of languages by SD. In this paper, we present a generalization of SD = AP to infinite words. This became possible via a substantial simplification of the proof for the corresponding result for finite words. Moreover, we show that SD = AP can be viewed as more fundamental than SF = AP in the sense that the classical 1965 result of Schützenberger and its 1980s extension to infinite words by Perrin are immediate consequences of SD = AP.

1 Introduction

A fundamental and classical result of Schützenberger [12] says that the class of star-free languages $\mathrm{SF}(A^*)$ is exactly the same as the class of aperiodic languages $\mathrm{AP}(A^*)$. This result was published in 1965 and later extended to infinite words by Perrin [9] in 1984. We simply write SF = AP when referring to this result. The class of star-free languages is very robust: it also coincides with the class $\mathrm{FO}[<]$ of languages definable in first-order logic [7]; and this is the same as the class LTL of languages definable in the linear temporal logic [6]. The equivalence of these characterizations have been established first for finite words and then extended to infinite words.

A proof for the equivalence can be conveniently arranged in a cycle. On this cycle the inclusion $\mathrm{AP} \subseteq \mathrm{LTL}$ becomes the most difficult one. It is done in the survey [3] with the concept of *local divisors* which play a prominent role here, too.

There is another beautiful characterization of $\mathrm{SF}(A^*)$ due to Schützenberger [13], which seems to be quite overlooked. It characterizes $\mathrm{SF}(A^*)$ without

* The second author gratefully acknowledges the support by the German Research Foundation (DFG) under grant DI 435/5-1 and the support by ANR 2010 BLAN 0202 FREC.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 89–98, 2012.

using complementation, but the inductive definition allows the star-operation on languages K (already belonging to the class) if K is a prefix code with bounded synchronization delay. Since synchronization delay is the main feature in this approach, the class is denoted by $\mathrm{SD}(A^*)$. The notion of bounded synchronization delay was introduced by Golomb and Gordon [5] and it is an important concept in coding theory. A proof of $\mathrm{SD}(A^*) = \mathrm{SF}(A^*)$ can be found e.g. in the textbook by Pin and Perrin on infinite words [10]. But although the book is on infinite words, this result is shown for finite words, only.

The extension to infinite words is actually done for the first time in this paper. This generalization became possible through the technique of local divisors, which also simplifies the classical proof on finite words. Our main result Theorem 1 is slightly more precise than simply stating SD = AP for infinite words: The extension of SD = AP to infinite words has a very nice explicit description where the ω-terms are just above star-free prefix codes of bounded synchronization delay. Moreover, we show that SD = AP can be viewed as more fundamental than SF = AP in the sense that the classical 1965 result of Schützenberger and its 1984 extension to infinite words by Perrin are immediate consequences of SD = AP.

We see therefore three contributions in this paper: (1) We considerably simplify the classical proof SD = AP by using the algebraic tool of a local divisor. (2) We easily extend SD = AP to infinite words by the very same proof technique. (3) We establish that SF = AP is an immediate consequence of SD = AP. This last property can also be seen as advertisement for the class SD.

2 Preliminaries

In the following A means a finite alphabet and A^* (resp. A^ω) denotes the sets of finite (resp. infinite) words over A. We let $A^\infty = A^* \cup A^\omega$. The empty word is denoted by 1.

A classical result in formal language theory says that a language $L \subseteq A^*$ is regular if and only if it is *recognizable*, see e.g. [11]. The latter means that there is a homomorphism $\varphi : A^* \to M$ to a finite monoid M such that $L = \varphi^{-1}\big(\varphi(L)\big)$. If φ is fixed and $u \in A^*$ is a word, we also write $[u]_\varphi$ instead of $\varphi^{-1}\big(\varphi(u)\big) = \{v \in A^* \mid \varphi(v) = \varphi(u)\}$. Thus, if $\varphi : A^* \to M$ recognizes L, then we can write L as a finite union $L = \bigcup \{[u]_\varphi \mid u \in L\}$.

A finite monoid M is called *aperiodic* if there exists some $n \in \mathbb{N}$ such that $x^n = x^{n+1}$ for all $x \in M$. Accordingly, a language $L \subseteq A^*$ is called *aperiodic* if it is recognized by some finite aperiodic monoid M. The class of aperiodic languages in A^* is denoted by $\mathrm{AP}(A^*)$.

Recognizability for ω-words is a little bit more technical to explain than for finite words. We give here a general definition for ∞-words, which applies to the finitary and to the infinitary case simultaneously, but keeps the ability to distinguish between the empty word, finite non-empty words, and infinite words, respectively.

Every word $u \in A^\infty$ can be written as either finite or infinite sequence $u = u_1u_2u_3\cdots$ with $u_i \in A^+$. We have $u_1u_2u_3\cdots \in A^*$ if and only if the sequence is finite, and $u = 1$ if the sequence is empty. The length of the sequence is an element in $\mathbb{N} \cup \{\omega\}$. Given a homomorphism $\varphi : A^* \to M$, we set

$$u_1u_2u_3\cdots \sim_\varphi v_1v_2v_3\cdots$$

if the two sequences have the same length and if $\varphi(u_i) = \varphi(v_i)$ for all i. A language $L \subseteq A^\infty$ is called *recognizable* or *regular* (resp. *aperiodic*) if there is a homomorphism $\varphi : A^* \to M$ to a finite (resp. finite and aperiodic) monoid M such that $u \in L$ and $u \sim_\varphi v$ implies $v \in L$ for all $u, v \in A^\infty$. If $L \subseteq A^\infty$ is regular (resp. aperiodic), then $L \cap A^*$ is regular (resp. aperiodic) in the usual sense. Moreover regular (resp. aperiodic) languages of A^* in the sense defined above remain regular (resp. aperiodic) in the new definition. As usual, $\mathrm{AP}(A^\infty)$ denotes the class of all aperiodic languages in A^∞. Similarly, $\mathrm{AP}(A^\omega)$ contains all aperiodic languages in A^ω. We remark that regular languages in A^ω are sometimes called *omega-regular* (or ω-regular). We shall use the following simple observation.

Lemma 1. *Let $L \subseteq A^\infty$ be regular. Then the following assertions are equivalent.*

1. $L \in \mathrm{AP}(A^\infty)$.
2. *There exists $n \in \mathbb{N}$ such that for all $u, x, y, z \in A^*$ we have*

$$\big(xu^n yz^\omega \in L \Leftrightarrow xu^{n+1}yz^\omega \in L\big) \quad \text{and} \quad \big(x(u^n y)^\omega \in L \Leftrightarrow x(u^{n+1}y)^\omega \in L\big).$$

3. *There exists $n \in \mathbb{N}$ such that for all $u, x, y, z \in A^*$ we have*

$$\big(xu^n yz^\omega \in L \Rightarrow xu^{n+1}yz^\omega \in L\big) \quad \text{and} \quad \big(x(u^n y)^\omega \in L \Rightarrow x(u^{n+1}y)^\omega \in L\big).$$

Proof. The language L is aperiodic if and only if its syntactic monoid in the sense of Arnold [1] is aperiodic, see e.g. [10]. Hence, 1 is equivalent to 2. The implication from 2 to 3 is trivial. It remains to show that 3 implies 2. It is here where we use the hypothesis that $L \subseteq A^\infty$ is regular. Since L is regular, its syntactic monoid is finite and it recognizes L. Hence there is some $p > 0$ such that, for all n large enough, $xu^n yz^\omega \in L$ if and only if $xu^{n+p}yz^\omega \in L$, and $x(u^n y)^\omega \in L$ if and only if $x(u^{n+p}y)^\omega \in L$. The result follows because by applying the hypothesis $p-1$ times, we may conclude that $xu^{n+1}yz^\omega \in L$ implies $xu^{n+p}yz^\omega \in L$ and that $x(u^{n+1}y)^\omega \in L$ implies $x(u^{n+p}y)^\omega \in L$. □

The language $(aa)^*$ is not aperiodic, but $(aa)^\omega = a^\omega$ is aperiodic. However we can infer aperiodicity of K^ω from that of K^*. The following lemma is well-known and it belongs to folklore. For lack of a precise reference and for convenience of the reader we give its proof.

Lemma 2. *If $K^* \in \mathrm{AP}(A^*)$, then $K^\omega \in \mathrm{AP}(A^\infty)$.*

Proof. Choose $m \in \mathbb{N}$ such that $xu^m\tilde{y} \in K^*$ implies $xu^{m+1}\tilde{y} \in K^*$ for all $u, x, \tilde{y} \in A^*$. Let $xu^myz^\omega \in K^\omega$. Then we find a prefix $xu^m\tilde{y} \in K^*$ such that $xu^myz^\omega = xu^m\tilde{y}\tilde{z}^\omega$. It follows that $xu^nyz^\omega \in K^\omega$ implies $xu^{n+1}yz^\omega \in K^\omega$ for all $n \geq m$. This is the first part in statement 3 of Lemma 1.

Now let $n = 2m$ and consider $x(u^ny)^\omega = v_1v_2v_3\cdots$ with $v_i \in K^+$ for all $i \in \mathbb{N}$. By Lemma 1 it remains to show that $x(u^{n+1}y)^\omega \in K^\omega$. We may assume $u \neq 1$. The infinite sequence $v_1v_2v_3\cdots$ defines cut points in the infinite word $x(u^ny)^\omega$ by choosing the positions between v_i and v_{i+1}. By gathering factors $v_j \cdots v_\ell$ together, we may assume that each factor u^n contains at most one cut point. We can write each $u^n = u^mu^m$ such that either the first u^m or the second u^m is without any cut point, and we can apply the hypothesis at all occurrences of u^n. Thus $x(u^{n+1}y)^\omega \in K^\omega$. □

3 Local Divisors

Let M be a finite monoid and let $c \in M$. Consider the submonoid $M' = \{x \in M \mid xc \in cM\}$. The right-translation by c shifts M' to $M'c = cM \cap Mc$. According to [2] we turn $cM \cap Mc$ into a monoid by defining a new multiplication $\circ$ on $cM \cap Mc$ by:

$$xc \circ cy = xcy.$$

It is straightforward to see that $\circ$ is well-defined and $(cM \cap Mc, \circ)$ is a monoid with neutral element c. Moreover, $x \mapsto xc$ yields a surjective homomorphism from M' onto $(cM \cap Mc, \circ)$. Thus, $(cM \cap Mc, \circ)$ becomes a divisor of M; it is called the *local divisor* of M at c. As a consequence, if M is aperiodic, then $(cM \cap Mc, \circ)$ is aperiodic, too. Local divisors were introduced in commutative algebra in [8]. In finite semigroup theory they first appear in [2] and have been used, among others, in a recent proof of the Krohn-Rhodes Theorem [4].

Lemma 3. *If M is aperiodic and $c \neq 1$, then $|cM \cap Mc| \leq |cM| < |M|$.*

Proof. Assume $1 \in cM$. Then there exists $d \in M$ such that $1 = cd$. Since M is aperiodic, we have $c^n = c^{n+1}$ for some $n \in \mathbb{N}$. Now, $1 = c \cdot 1 \cdot d = c^nd^n = c^{n+1}d^n = c$, a contradiction. Thus $1 \in M \setminus cM$. □

4 Schützenberger's Class SD

A language $K \subseteq A^*$ is called *prefix-free* if $u, uv \in K$ implies $u = uv$. A prefix-free language $K \subseteq A^+$ is also called a *prefix code* since every word $u \in K^*$ admits a unique factorization $u = u_1 \cdots u_k$ with $k \geq 0$ and $u_i \in K$. A prefix code K has *bounded synchronization delay* if for some $d \in \mathbb{N}$ and for all $u, v, w \in A^*$ we have:

$$\text{if } uvw \in K^* \text{ and } v \in K^d, \text{ then } uv \in K^*.$$

If d is given explicitly, we also say that K has synchronization delay d. Note that every subset $B \subseteq A$ yields a prefix code with synchronization delay 0. In

particular, the sets B are prefix codes of bounded synchronization delay for all $B \subseteq A$.

The intuition behind this concept is the following: Assume a sender emits a stream of code words from K, where K is a prefix code of bounded synchronization delay d. If a receiver misses the beginning of the message, he can wait until he detects a sequence of d code words. Then he can synchronize and decipher the remaining text after these d words.

We now inductively define Schützenberger's language class $\mathrm{SD}(A^\infty)$ simultaneously for finite and infinite words as follows:

1. We have $\emptyset \in \mathrm{SD}(A^\infty)$ and $\{a\} \in \mathrm{SD}(A^\infty)$ for all letters $a \in A$.
2. If $L, K \in \mathrm{SD}(A^\infty)$, then $L \cup K \in \mathrm{SD}(A^\infty)$.
3. If $L, K \in \mathrm{SD}(A^\infty)$, then $(L \cap A^*) \cdot K \in \mathrm{SD}(A^\infty)$.
4. If $K \in \mathrm{SD}(A^\infty)$ with $K \subseteq A^+$ such that K is a prefix code with bounded synchronization delay, then $K^*, K^\omega \in \mathrm{SD}(A^\infty)$.

Note that (unlike the definition of star-free sets) the inductive definition of $\mathrm{SD}(A^\infty)$ does not use any complementation. The class $\mathrm{SF}(A^\infty)$ of star-free languages is defined similarly, but the closure property 4 is replaced by

4'. If $K \in \mathrm{SF}(A^\infty)$, then $A^\infty \setminus K \in \mathrm{SF}(A^\infty)$.

By $\mathrm{SD}(A^*)$ we denote the finitary subclass of $\mathrm{SD}(A^\infty)$. That is, $\mathrm{SD}(A^*) = \{L \in \mathrm{SD}(A^\infty) \mid L \subseteq A^*\}$. Similarly, $\mathrm{SD}(A^\omega)$ is its infinitary subclass. Also note that $\emptyset^* = \{1\}$ belongs to $\mathrm{SD}(A^*)$ because $\emptyset$ is a prefix code of bounded synchronization delay.

Example 1. Let $B \subseteq A$. For every letter $c \in B$ we let $B_c = B \setminus \{c\}$. Then $B_c^* c \in \mathrm{SD}(A^\infty)$ and $B_c^* c$ is a prefix code with synchronization delay 1. Thus the language $L_c = (B_c^* c)^\omega$ of all words with infinitely many c is in $\mathrm{SD}(A^\infty)$. Since every infinite word contains some letter occurring infinitely often, we have $B^\omega = \bigcup_{c \in B} L_c$. In particular, the set of all infinite words B^ω over the alphabet B is in $\mathrm{SD}(A^\infty)$. ◇

Lemma 4. *We have* $\mathrm{SD}(A^\infty) \subseteq \mathrm{AP}(A^\infty)$.

Proof. The class $\mathrm{AP}(A^\infty)$ contains all finite subsets of A^∞ and it is closed under finite union and concatenation, see e.g. [3]. By Lemma 2 it is enough to show the following claim which concerns only finitary languages: If $K \in \mathrm{SD}(A^*)$ is a prefix code with synchronization delay d, then K^* is aperiodic. By induction we may assume that $K \in \mathrm{AP}(A^*)$. Hence, by Lemma 1, for some $n \in \mathbb{N}$ and all words u, x, y we have $xu^n y \in K$ if and only if $xu^{n+1}y \in K$. Moreover, by the same lemma it is enough to show that $xu^{n(d+1)}y \in K^*$ implies $xu^{n(d+1)+1}y \in K^*$ for all words u, x, y.

Consider $u \in A^+$ and let $m = n(d+1)$. Suppose $xu^m y \in K^+$. There is a unique factorization $xu^m y = v_1 \cdots v_k$ with $v_i \in K$. If u^n occurs as a factor of some v_i, then we are done, because inside such a factor v_i we can replace u^n by u^{n+1}. Therefore, we can assume $xu_1u_2 \in K^+$ for $u_1u_2u_3 = u^{m-1}$ with $u_2 \in K^d$.

Since K is a prefix code, we have $q = u_3uy \in K^*$. The prefix $p = xuu_1u_2$ of xu^my ends with a word in K^d. Note the extra u after x in p. Thus, by synchronization delay, we conclude $p \in K^*$. It follows that $xu^{m+1}y = pq \in K^+$. □

5 SD Equals AP

We consider the general situation for languages in A^∞, i.e., we deal with finite and infinite words simultaneously. Theorem 1 is our main contribution. The first part of the proof also yields a simple proof for the corresponding result $\mathrm{SD}(A^*) = \mathrm{AP}(A^*)$ over finite words.

Theorem 1. *Let $L \subseteq A^\infty$. The following assertions are equivalent:*

1. $L \in \mathrm{SD}(A^\infty)$.
2. $L \in \mathrm{AP}(A^\infty)$.
3. *L can be written as finite union*

$$L \;=\; L_0 \cup \bigcup_{i=1}^{m} L_i\, K_i^\omega,$$

where L_i, $K_i \in \mathrm{SD}(A^*)$ and all K_i are prefix codes with bounded synchronization delay.

Proof. Recall that if $K \in \mathrm{SD}(A^*)$ is a prefix code of bounded synchronization delay, then $K^\omega \in \mathrm{SD}(A^\infty)$ by definition. The implication $3 \Rightarrow 1$ is therefore trivial. The implication $1 \Rightarrow 2$ is the content of Lemma 4. It remains to show the implication $2 \Rightarrow 3$. Note that for $L \subseteq A^*$ this means just to show $2 \Rightarrow 1$.

We start with a recognizing homomorphism $\varphi : A^* \to M$ to a finite aperiodic monoid. We will also show that the languages L_i and K_i appearing in the expression given by 3 can be chosen from a finite collection of sets which depends on the homomorphism φ, only.

Let us denote by $\approx_\varphi$ the equivalence relation generated on A^∞ by $\sim_\varphi$, and for each $w \in A^\infty$ let $[\![w]\!]_\varphi = \{v \in A^\infty \mid v \approx_\varphi w\}$. Since L is regular, there are only finitely many classes $[\![w]\!]_\varphi$. Note that according to our definition of $\sim_\varphi$ and $\approx_\varphi$ we have three possibilities: $[\![w]\!]_\varphi = \{1\}$ or $[\![w]\!]_\varphi = [w]_\varphi \setminus \{1\} \subseteq A^+$ or $[\![w]\!]_\varphi \subseteq A^\omega$.

We show that for every word $w \in A^\infty$ there exists a language $L(w) \in \mathrm{SD}(A^\infty)$ satisfying $w \in L(w) \subseteq [\![w]\!]_\varphi$ such that the size of the rational expression for $L(w)$ is bounded by a function in $|M|$ and $|A|$; in particular, it does not depend on the length of w. In addition, if w is a finite word, then the expression for $L(w)$ does not use terms of the form K^ω. It follows $L = \bigcup_{w \in L} L(w)$ and this union is finite since there are only finitely many languages $L(w)$. The proof is by induction on the parameter $(|M|, |A|)$ with lexicographic order.

Let $C = \{a \in A \mid a \text{ occurs in } w\}$. First, suppose $|\varphi(C^*)| = 1$. If $w = 1$, we have $L(w) = \{1\}$; if $w \in A^+$, then we can set $L(w) = C^+$; and if $w \in A^\omega$, then $L(w) = C^\omega$ and this language is in $\mathrm{SD}(A^\infty)$ by Example 1. This covers both cases $|M| = 1$ and $A = \emptyset$. Hence, in the remainder of the proof we may assume $\varphi(c) \neq 1$

for some letter $c \in A$ occurring in w. Let $B = A \setminus \{c\}$. We write $w = uv$ with $u \in B^*$ and $v \in cA^\infty$. By induction on the alphabet, there exists $L(u) \in \mathrm{SD}(B^*)$ and it remains to construct $L(v) \in \mathrm{SD}(A^\infty)$. Then $L(w) = L(u)L(v)$. In the remainder of the proof we therefore assume $w \in cA^\infty$. We now distinguish two cases: The word w contains only finitely many occurrences of the letter c or it contains infinitely many c.

5.1 Finitely Many c

We can write $w = uw'$ with $u \in c(B^*c)^*$ and $w' \in B^\infty$. By induction, there exists $L(w') \in \mathrm{SD}(B^\infty)$ and it suffices to construct $L(u) \in \mathrm{SD}(A^*)$. Define a new finite alphabet $T = \{[v]_\varphi \mid v \in B^*\}$ and let $M_c = (\varphi(c)M \cap M\varphi(c), \circ)$ denote the local divisor of M at $\varphi(c)$, see Section 3. Let $\psi : T^* \to M_c$ be the homomorphism induced by the mapping $[v]_\varphi \mapsto \varphi(cvc)$ for $[v]_\varphi \in T$. Next, we define a mapping (*substitution*) $\sigma : (B^*c)^* \to T^*$ by $\sigma(v_1 c \cdots v_k c) = [v_1]_\varphi \cdots [v_k]_\varphi$ for $v_i \in B^*$. By definition of the operation $\circ$ in M_c we see:

$$\begin{aligned}\psi(\sigma(v_1 c \cdots v_k c)) &= \psi([v_1]_\varphi \cdots [v_k]_\varphi) \\ &= \psi([v_1]_\varphi) \circ \cdots \circ \psi([v_k]_\varphi) \\ &= \varphi(cv_1 c \cdots v_k c).\end{aligned}$$

Hence, for all $m \in M_c$ we obtain $\varphi^{-1}(m) \cap c(B^*c)^* = c\,\sigma^{-1}(\psi^{-1}(m))$. Thus, we can set $L(u) = c\,\sigma^{-1}\big(L(\sigma(u'))\big)$ for $u = cu'$. By Lemma 3, we have $|M_c| < |M|$ and therefore, by induction on the size of the monoid, $L(\sigma(u')) \in \mathrm{SD}(T^*)$ exists. It remains to show that $K \in \mathrm{SD}(T^\infty)$ implies $\sigma^{-1}(K) \in \mathrm{SD}(A^\infty)$. This last step is done by structural induction over the expression for K. For every $v \in B^*$ there exists a language $L(v) \in \mathrm{SD}(B^*)$ by induction on the size of the alphabet. Hence, for every letter $t \in T$ we have:

$$\sigma^{-1}(t) = \left(\bigcup_{v \in B^*,\ [v]_\varphi = t} L(v) \right) c$$

and this union is finite. For $L, K \in \mathrm{SD}(T^*)$ we have:

$$\begin{aligned}\sigma^{-1}(L \cup K) &= \sigma^{-1}(L) \cup \sigma^{-1}(K), \\ \sigma^{-1}(L\,K) &= \sigma^{-1}(L)\,\sigma^{-1}(K), \\ \sigma^{-1}(K^*) &= \sigma^{-1}(K)^*, \\ \sigma^{-1}(K^\omega) &= \sigma^{-1}(K)^\omega.\end{aligned}$$

We still have to see that $\sigma^{-1}(K)$ is a prefix code of bounded synchronization delay, if K has this property. Clearly, $1 \notin \sigma^{-1}(K)$. To see prefix-freeness, consider $u, uv \in \sigma^{-1}(K)$. This implies $u \in A^*c$ and hence, $\sigma(uv) = \sigma(u)\sigma(v)$. It follows that $v = 1$ because K is prefix-free. Finally, let $L = \sigma^{-1}(K)$ and suppose that K has synchronization delay d. We show that L has synchronization delay $d+1$: Let $uvw \in L^*$ with $v \in L^{d+1}$. Write $v = u'cv'$ with $v' \in L^d$. Note that $v' \in A^*c$.

It follows $\sigma(uv) = \sigma(uu'c)\sigma(v')$ and $\sigma(v') \in K^d$. Thus, $\sigma(uv) \in K^*$. We obtain $uv \in L^*$ as desired. This completes the case where there are only finitely many c. In particular, we can derive $\mathrm{SD}(A^*) = \mathrm{AP}(A^*)$ at this point.

5.2 Infinitely Many c

We have $w = cw'$ with $w' \in (B^*c)^\omega$. The substitution σ is extended from finite to infinite sequences $\sigma : (B^*c)^\omega \to T^\omega$ by $\sigma(v_1cv_2c\cdots) = [v_1]_\varphi[v_2]_\varphi \cdots$ for $v_i \in B^*$. By induction on the size of the monoid, there are languages L_T, $K_T \in \mathrm{SD}(T^*)$ such that K_T is a prefix code of bounded synchronization delay, which satisfy:

$$\sigma(w') \in L_T\, K_T^\omega \subseteq [\![\sigma(w')]\!]_\psi\,.$$

The languages L_T and K_T can be chosen from a finite set depending on the homomorphism $\psi : T^* \to M_c$, only. Let $L = \sigma^{-1}(L_T)$ and $K = \sigma^{-1}(K_T)$. By Section 5.1 we know that L, $K \in \mathrm{SD}(A^*)$ such that K is a prefix code of bounded synchronization delay. Let $L(w) = cLK^\omega$; then we have $w \in L(w)$. We still have to prove $cLK^\omega \subseteq [\![cw]\!]_\varphi$. This is a subtle point. It is achieved by some trick as done in [2].

Let $v \in cLK^\omega$. Then we have $\sigma(v') \in [\![\sigma(w')]\!]_\psi$ for $v = cv'$ and $w = cw'$. Hence $\sigma(v') \approx_\psi \sigma(w')$. Since $\approx_\psi$ is the equivalence relation generated by $\sim_\psi$, it remains to show the following claim.

Claim: *$\sigma(v) \sim_\psi \sigma(w)$ implies $cv \approx_\varphi cw$ for all $v, w \in (B^*c)^\omega$.*

To see the claim write $\sigma(v) = \sigma(v_1c)\sigma(v_2c)\cdots$ and $\sigma(w) = \sigma(w_1c)\sigma(w_2c)\cdots$ with v_ic, $w_ic \in (B^*c)^+$ such that $\psi(\sigma(v_ic)) = \psi(\sigma(w_ic))$ for all $i \in \mathbb{N}$. It follows $\varphi(cv_ic) = \varphi(cw_ic)$ for all $i \in \mathbb{N}$, see Section 5.1. Thus

$$\begin{aligned} cv = {} & (cv_1c)v_2(cv_3c)v_4(c\cdots \\ \sim_\varphi {} & (cw_1c)v_2(cw_3c)v_4(c\cdots \\ = {} & cw_1(cv_2c)w_3(cv_4c)\cdots \\ \sim_\varphi {} & cw_1(cw_2)cw_3(cw_4c)\cdots \\ = {} & cw. \end{aligned}$$

This completes the proof of the claim and the proof of the theorem. □

Corollary 1. *We have $\mathrm{SD}(A^*) = \mathrm{AP}(A^*)$ and $\mathrm{SD}(A^\omega) = \mathrm{AP}(A^\omega)$. Moreover, every language $\mathrm{SD}(A^\omega)$ is a finite union of languages of the form LK^ω where $L, K \in \mathrm{SD}(A^*)$ and K is a prefix code with bounded synchronization delay.*

Proof. The first two statements immediately follow from $\mathrm{SD}(A^\infty) = \mathrm{AP}(A^\infty)$ in Theorem 1. The last property follows by taking $L_0 = \emptyset$ in statement 3 of the same theorem. □

6 SD = AP Implies SF = AP

The aim of this section is to show that Theorem 1 can be viewed as more general than the classic result that star-freeness is equivalent to aperiodicity. In this sense we would like to propose the thesis that SD = AP is a more fundamental result in formal language theory than the celebrated result SF = AP. We are aware that such a thesis is debatable. Therefore, we spend a few words to explain the idea.

First, by Theorem 1 we have $\mathrm{SD}(A^\infty) = \mathrm{AP}(A^\infty)$. Hence, $\mathrm{SD}(A^\infty)$ is closed under complementation and therefore, trivially, $\mathrm{SF}(A^\infty) \subseteq \mathrm{SD}(A^\infty)$. Thus, in order to establish $\mathrm{SD}(A^\infty) = \mathrm{SF}(A^\infty) = \mathrm{AP}(A^\infty)$ it remains to see that $\mathrm{SD}(A^\infty) \subseteq \mathrm{SF}(A^\infty)$. For this inclusion in turn, it is enough to prove the following simple fact.

Lemma 5. *If $K \in \mathrm{SF}(A^*)$ is a prefix code of bounded synchronization delay, then $A^\infty \setminus K^\infty$ is star-free.*

Proof. As K is a prefix code we can write $A^\infty \setminus K^\infty$ as an infinite union:

$$A^\infty \setminus K^\infty = \bigcup_{0 \le i} \left(K^i A A^\infty \setminus K^{i+1} A^\infty \right). \tag{1}$$

Now, let d be the synchronization delay of K. Then we can write

$$A^\infty \setminus K^\infty = A^* K^d (A A^\infty \setminus K A^\infty) \cup \bigcup_{0 \le i < d} (K^i A A^\infty \setminus K^{i+1} A^\infty).$$

The inclusion from left to right follows from Equation (1). The other inclusion holds since the intersection of K^∞ and $A^* K^d (A A^\infty \setminus K A^\infty)$ is empty. This is obtained by using the definition of synchronization delay. □

Several comments are adequate.

Remark 1. The result of Lemma 5 for finitary languages is due to Schützenberger [13]. Another proof (which yields a star-free expression for $A^* \setminus K^*$) can be found in the textbook of Perrin and Pin [10, Lemma VIII.6.5]. ◇

Remark 2. The proof for $\mathrm{SD}(A^\infty) = \mathrm{SF}(A^\infty) = \mathrm{AP}(A^\infty)$ can be arranged in a cycle. In this case we would start with Lemma 5 showing directly that $\mathrm{SD}(A^\infty) \subseteq \mathrm{SF}(A^\infty)$. Next, one uses the classical construction showing that $\mathrm{AP}(A^\infty)$ is closed under concatenation. This yields $\mathrm{SF}(A^\infty) \subseteq \mathrm{AP}(A^\infty)$. Now, the final and most difficult step is just the implication from 2 to 3 in Theorem 1. This concludes the cycle. Note that in this cycle the statement of Lemma 4 has not been used because in a cycle it can be substituted by Lemma 5. ◇

Remark 3. We have just argued that $\mathrm{SF}(A^\infty) = \mathrm{AP}(A^\infty)$ is an immediate consequence of the result $\mathrm{SD}(A^\infty) = \mathrm{AP}(A^\infty)$. We claim that it is however not equally simple to deduce $\mathrm{SD}(A^\infty) = \mathrm{AP}(A^\infty)$ from $\mathrm{SF}(A^\infty) = \mathrm{AP}(A^\infty)$. Indeed, the easy part is to see that $\mathrm{SD}(A^\infty) \subseteq \mathrm{AP}(A^\infty)$ or $\mathrm{SD}(A^\infty) \subseteq \mathrm{SF}(A^\infty)$. But then it remains the hard part which is to show one of reverse inclusions. ◇

Acknowledgments. We would like to thank Jean-Éric Pin for bringing the class SD to our attention and for the proposal that the notion of *local divisor* might lead to a simplified proof for $\mathrm{SD}(A^*) = \mathrm{AP}(A^*)$.

References

1. Arnold, A.: A syntactic congruence for rational ω-languages. Theoretical Computer Science 39, 333–335 (1985)
2. Diekert, V., Gastin, P.: Pure future local temporal logics are expressively complete for Mazurkiewicz traces. Information and Computation 204, 1597–1619 (2006); Conference version in LATIN 2004. LNCS, vol. 2976, pp. 170–182 (2004)
3. Diekert, V., Gastin, P.: First-order definable languages. In: Logic and Automata: History and Perspectives. Texts in Logic and Games, pp. 261–306. Amsterdam University Press (2008)
4. Diekert, V., Kufleitner, M., Steinberg, B.: The Krohn-Rhodes Theorem and local divisors. arXiv, abs/1111.1585 (2011)
5. Golomb, S.W., Gordon, B.: Codes with bounded synchronization delay. Information and Control 8(4), 355–372 (1965)
6. Kamp, J.A.W.: Tense Logic and the Theory of Linear Order. PhD thesis, University of California, Los Angeles, California (1968)
7. McNaughton, R., Papert, S.: Counter-Free Automata. The MIT Press (1971)
8. Meyberg, K.: Lectures on algebras and triple systems. Technical report, University of Virginia, Charlottesville (1972)
9. Perrin, D.: Recent Results on Automata and Infinite Words. In: Chytil, M.P., Koubek, V. (eds.) MFCS 1984. LNCS, vol. 176, pp. 134–148. Springer, Heidelberg (1984)
10. Perrin, D., Pin, J.-É.: Infinite words. Pure and Applied Mathematics, vol. 141. Elsevier, Amsterdam (2004)
11. Pin, J.-É.: Varieties of Formal Languages. North Oxford Academic, London (1986)
12. Schützenberger, M.P.: On finite monoids having only trivial subgroups. Inf. Control 8, 190–194 (1965)
13. Schützenberger, M.P.: Sur certaines opérations de fermeture dans les langages rationnels. In: Symposia Mathematica, Convegno di Informatica Teorica, INDAM, Roma, 1973, vol. XV, pp. 245–253. Academic Press, London (1975)

Towards Optimal Degree-Distributions for Left-Perfect Matchings in Random Bipartite Graphs*

Martin Dietzfelbinger and Michael Rink

Fakultät für Informatik und Automatisierung, Technische Universität Ilmenau
{martin.dietzfelbinger,michael.rink}@tu-ilmenau.de

Abstract. Consider a random bipartite multigraph G with n left nodes and $m \geq n \geq 2$ right nodes. Each left node x has $d_x \geq 1$ random right neighbors. The average left degree $\bar{\Delta}$ is fixed, $\bar{\Delta} \geq 2$. We ask whether for the probability that G has a left-perfect matching it is advantageous not to fix d_x for each left node x but rather choose it at random according to some (cleverly chosen) distribution. We show the following, provided that the degrees of the left nodes are independent: If $\bar{\Delta}$ is an integer then it is optimal to use a fixed degree of $\bar{\Delta}$ for all left nodes. If $\bar{\Delta}$ is non-integral then an optimal degree-distribution has the property that each left node x has two possible degrees, $\lfloor \bar{\Delta} \rfloor$ and $\lceil \bar{\Delta} \rceil$, with probability p_x and $1 - p_x$, respectively, where p_x is from the closed interval $[0, 1]$ and the average over all p_x equals $\lceil \bar{\Delta} \rceil - \bar{\Delta}$. Furthermore, if $n = c \cdot m$ and $\bar{\Delta} > 2$ is constant, then each distribution of the left degrees that meets the conditions above determines the same threshold $c^*(\bar{\Delta})$ that has the following property as n goes to infinity: If $c < c^*(\bar{\Delta})$ then there exists a left-perfect matching with high probability. If $c > c^*(\bar{\Delta})$ then there exists no left-perfect matching with high probability. The threshold $c^*(\bar{\Delta})$ is the same as the known threshold for offline k-ary cuckoo hashing for integral or non-integral $k = \bar{\Delta}$.

1 Introduction

We study bipartite multigraphs G with left node set S and right node set T, where each left node x from S has D_x right neighbors. The right neighbors are chosen at random with replacement from T, where the number of choices D_x is a random variable that follows some probability mass function ρ_x. Let $|S| = n$ and let $|T| = m$ as well as $1 \leq D_x \leq m$ for all x from S. For each x from S let Δ_x be the mean of D_x, that is, $\Delta_x = \sum_{l=1}^{m} l \cdot \rho_x(l)$, and let $\bar{\Delta}$ be the average mean, i.e., $\bar{\Delta} = 1/n \cdot \sum_{x \in S} \Delta_x$. We assume that the random variables $D_x, x \in S$, are independent and $\bar{\Delta}$ is a given constant.

Our aim is to determine a sequence of probability mass functions $(\rho_x)_{x \in S}$ for the random variables $(D_x)_{x \in S}$ that maximizes the probability that the random

* Research supported by DFG grant DI 412/10-2.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 99–111, 2012.

graph $G = G\big(\bar{\Delta}, (\rho_x)_{x\in S}\big)$ has a matching that covers all left nodes, i.e., a left-perfect matching[1]. We call such a sequence *optimal*. Note that there must be some optimal sequence for compactness reasons.

1.1 Motivation and Related Work

Studying irregular bipartite graphs has lead to major improvements in the performance of erasure correcting codes. For example in [6] Luby et al. showed how to increase the fraction of message bits that can be recovered for a fixed number of check bits by using carefully chosen degree sequences for both sides of the underlying bipartite graph. The recovery process for erased message bits translates directly into a greedy algorithm for finding a matching in the bipartite graph associated with the recovery process. This was the motivation for the authors of [1,2] to study irregularity in the context of offline k-ary cuckoo hashing. Here one has a bipartite graph with left nodes corresponding to keys and right nodes corresponding to table cells, where each key randomly chooses table cells without replacement and the aim is essentially to find a left-perfect matching. In [1] it was proven that if the degree of each left node follows some distribution with identical mean and is independent of the other nodes then it is optimal in an asymptotic sense if the degree of each left node is concentrated around its mean. This is in contrast of the following observation in [7] in analogy to [6]: an uneven distribution of the degrees of the left nodes can increase the probability for the existence of a matching that has the advantage that it can be calculated in linear time, by successively assigning left nodes to right nodes of degree one and removing them from the graph.

1.2 Results

We will show that for given parameters n, m, and $\bar{\Delta}$ there is an optimal sequence of probability mass functions that concentrates the degree of the left nodes around $\lfloor\bar{\Delta}\rfloor$ and $\lceil\bar{\Delta}\rceil$. Furthermore, if $\bar{\Delta}$ is an integer we can explicitly determine this optimal sequence. In the case that $\bar{\Delta}$ is non-integral we will identify a tight condition that an optimal sequence must meet.

Theorem 1. *Let $n \le m$, as well as $n, \bar{\Delta} \ge 2$, and let $(\rho_x)_{x\in S}$ be an optimal sequence for parameters $(n, m, \bar{\Delta})$. Then the following holds for all $x \in S$.*
(i) If $\bar{\Delta}$ is an integer, then $\rho_x(\bar{\Delta}) = 1$.
(ii) If $\bar{\Delta}$ is non-integral, then $\rho_x(\lfloor\bar{\Delta}\rfloor) \in [0, 1]$ and $\rho_x(\lceil\bar{\Delta}\rceil) = 1 - \rho_x(\lfloor\bar{\Delta}\rfloor)$.

The second statement is not entirely satisfying since it identifies no optimal solution. However, we will give strong evidence that in the situation of Theorem 1 (*ii*) there is no single, simple description of a distribution that is optimal for all feasible node set sizes.

Since the case $\bar{\Delta} = 2$ is completely settled by Theorem 1 (*i*), we focus on the cases where $\bar{\Delta} > 2$, with the additional condition that the number of left nodes

[1] In the following we will use "matching" and "left-perfect matching" synonymously.

is linear in the number of right nodes, that is $n = c \cdot m$ for constant $c > 0$. We show that for sufficiently large n all sequences that meet the condition of Theorem 1 (*ii*) asymptotically lead to the same matching probability. Therefore, we call these sequences *near optimal*.

Proposition 1. *Let $n = c \cdot m$, for constant $c > 0$, and let $(\rho_x)_{x \in S}$ be a near optimal sequence with average expected degree $\bar{\Delta} > 2$. Then for sufficiently large n there is a threshold $c^*(\bar{\Delta})$ such that the random graph $G = G\big(\bar{\Delta}, (\rho_x)_{x \in S}\big)$ has the following property.*
(i) If $c < c^*(\bar{\Delta})$, then G has a matching with probability $1 - o(1)$.
(ii) If $c > c^*(\bar{\Delta})$, then G has no matching with probability $1 - o(1)$.

The threshold $c^*(\bar{\Delta})$ is exactly the same as the threshold given in the context of k-ary cuckoo hashing for integral k [5,4,2], and non-integral k [2], where $k = \bar{\Delta}$.

So in the case that $n = c \cdot m$ all near optimal sequences are hardly distinguishable in terms of matching probability, at least asymptotically, but we will give strong evidence that there are only two sequences that can be optimal, where the decision which one is the optimal one depends on the ratio c.

Conjecture 1. Let $(\rho_x)_{x \in S}$ be an *optimal sequence* for parameters $(n, m, \bar{\Delta})$ in the situation of Theorem 1 (*ii*) for $n = c \cdot m$ and constant $c > 0$ and $\bar{\Delta} > 2$. Let $\alpha = \lceil \bar{\Delta} \rceil - \bar{\Delta}$.
(*i*) If $c < c^*(\bar{\Delta})$, then $\rho_x(\lfloor \bar{\Delta} \rfloor) = 1$ for $\alpha \cdot n$ nodes and $\rho_x(\lceil \bar{\Delta} \rceil) = 1$ for $(1 - \alpha) \cdot n$ nodes (assuming that $\alpha \cdot n$ is an integer).
(*ii*) If $c > c^*(\bar{\Delta})$, then $\rho_x(\lfloor \bar{\Delta} \rfloor) = \alpha$ and $\rho_x(\lceil \bar{\Delta} \rceil) = 1 - \alpha$ for all $x \in S$.

That is, if c is to the left of the threshold then it is optimal to fix the degrees of the left nodes, and if c is to the right of the threshold then it is optimal to let each left node choose its degree at random from $\lfloor \bar{\Delta} \rfloor$ and $\lceil \bar{\Delta} \rceil$, by identical, independent experiments.

Overview of the Paper. The next section, which is also the main part, covers the proof of Theorem 1. It is followed by a section devoted to the discussion of Conjecture 1. The proof of Proposition 1 is given in the full version of this paper [3, Appendix B]. It uses standard techniques on concentration bounds for nodes of certain degrees.

2 Optimality of Concentration in a Unit Length Interval

In this section we prove Theorem 1. We define the *success probability* of a random graph as the probability that this graph has a matching. Let n, m and $\bar{\Delta}$ be fixed and consider some arbitrary but fixed sequence of probability mass functions $(\rho_x)_{x \in S}$. We will show that if this sequence has certain properties then we can do a modification, obtaining a new sequence $(\rho'_x)_{x \in S}$ with the same average expected value $\bar{\Delta}$, such that $G\big(\bar{\Delta}, (\rho'_x)_{x \in S}\big)$ has a strictly higher success probability than $G\big(\bar{\Delta}, (\rho_x)_{x \in S}\big)$.

Lemma 1 (Variant of [2, Proposition 4]). *Let $(\rho_x)_{x \in S}$ be given. Let $z \in S$ be arbitrary but fixed. If in ρ_z two degrees with distance at least 2 have nonzero probability then $(\rho_x)_{x \in S}$ is not optimal.*

The lemma was stated in [2] and proven in [1] for a slightly different graph model. Its proof runs along the lines of [1]; it is given in [3, Appendix A]. After applying the first lemma repeatedly one sees that in an optimal sequence each left node node has either a fixed degree (with probability 1) or two possible degrees with non-zero probability, where these degrees differ by 1. The lemma and [1,2] do not say anything about the relation between the degrees of different nodes. This follows next.

Lemma 2. *Let $(\rho_x)_{x\in S}$ be given, where for each $x \in S$ the only degrees with nonzero probability are from $\{\lfloor \Delta_x \rfloor, \lceil \Delta_x \rceil\}$. Let $y, z \in S$ be arbitrary but fixed. If $\lfloor \Delta_y \rfloor$ and $\lfloor \Delta_z \rfloor$ have distance at least 2, or $\lceil \Delta_y \rceil$ and $\lceil \Delta_z \rceil$ have distance at least 2, then $(\rho_x)_{x\in S}$ is not optimal.*

Lemma 2 is proved in Section 2.1. Using Lemma 2 one concludes that an optimal sequence restricts the means Δ_x, for each $x \in S$, to an open interval $(l-1, l+1)$ for some integer constant $l \geq 2$. Hence all degrees that appear with non-zero probability must be from $\{l-1, l, l+1\}$. With the help of the next lemma one concludes that actually two values are enough.

Lemma 3. *Let $(\rho_x)_{x\in S}$ be given, where for each $x \in S$ the only degrees with nonzero probability are from $\{\lfloor \Delta_x \rfloor, \lceil \Delta_x \rceil\}$. Let $y, z \in S$ be arbitrary but fixed and assume that Δ_y and Δ_z are non-integral. If $\lceil \Delta_y \rceil$ and $\lfloor \Delta_z \rfloor$ have distance 2 then $(\rho_x)_{x\in S}$ is not optimal.*

Lemma 3 is proved in Section 2.2. Combining Lemmas 1, 2, and 3, we obtain the following for an optimal sequence. If $l \leq \bar{\Delta} < l+1$ then it holds $l \leq \Delta_x \leq l+1$, for all $x \in S$, and all degrees that appear with non-zero probability must be from $\{l, l+1\}$. If $\bar{\Delta}$ is an integer, then by definition of $\bar{\Delta}$, we have $\rho_x(\bar{\Delta}) = 1$ for all $x \in S$. Hence Theorem 1 follows.

So, to complete the proof of the theorem, it remains to show the three lemmas, which is done in the following two sections for Lemmas 2 and 3, and in [3, Appendix A] for Lemma 1. We make use of the following definitions.

For each set $S' \subseteq S$ let $G_{S'}$ be the induced bipartite subgraph of G with left node set S' and right node set T, particularly $G_S = G$. A matching in $G_{S'}$ is a matching that covers all left nodes (left-perfect matching). We define $\mathcal{M}_{S'}$ as the event that $G_{S'}$ has a matching.

2.1 Average Degrees of Different Nodes are Close

In this section we prove Lemma 2. Consider the probability mass functions ρ_y and ρ_z for the degrees D_y and D_z respectively. By the hypothesis of the lemma, ρ_y and ρ_z are concentrated on two values each, i.e.,

$$\rho_y(k) = p,\ \rho_y(k+1) = 1-p \qquad \rho_z(l) = q,\ \rho_z(l+1) = 1-q\ ,$$

with $p \in [0,1]$ and $q \in [0,1]$. By the assumption, we may arrange things so that $k - l \geq 2$ and
(*i*) $k = \lfloor \Delta_y \rfloor$, $l = \lfloor \Delta_z \rfloor$ as well as $p = 1 - (\Delta_y - \lfloor \Delta_y \rfloor)$, $q = 1 - (\Delta_z - \lfloor \Delta_z \rfloor)$,

or (ii) $k+1 = \lceil \Delta_y \rceil$, $l+1 = \lceil \Delta_z \rceil$ as well as $p = \lceil \Delta_y \rceil - \Delta_y$, $q = \lceil \Delta_z \rceil - \Delta_z$. We will show that changing ρ_y to ρ'_y and ρ_z to ρ'_z such that $\Delta'_y = \Delta_y - 1$ and $\Delta'_z = \Delta_z + 1$, via

$$\rho'_y(k-1) = p,\ \rho'_y(k) = 1-p \qquad \rho'_z(l+1) = q,\ \rho'_z(l+2) = 1-q\ ,$$

will strictly increase the probability that G_S has a matching, while it does not change $\bar{\Delta}$. For this, will show

$$\Pr\left(\bar{\mathcal{M}}_S \mid \rho_y, \rho_z\right) > \Pr\left(\bar{\mathcal{M}}_S \mid \rho'_y, \rho'_z\right)\ ,$$

abusing condition notation a little to indicate changed probability spaces. We fix the neighborhood N_x for the remaining elements $x \in S - \{y,z\}$ and therefore the graph $G_{S-\{y,z\}}$. Since there can be a matching for S only if there is a matching for $S - \{y,z\}$ it is sufficient to show that

$$\Pr\left(\bar{\mathcal{M}}_S \mid \mathcal{M}_{S-\{y,z\}}, \rho_y, \rho_z\right) > \Pr\left(\bar{\mathcal{M}}_S \mid \mathcal{M}_{S-\{y,z\}}, \rho'_y, \rho'_z\right)\ . \tag{1}$$

Let $\mathrm{Fail}(d_y, d_z) = \Pr\left(\bar{\mathcal{M}} \mid \mathcal{M}_{S-\{y,z\}}, D_y = d_y, D_z = d_z\right)$. Then (1) holds if and only if

$$\sum_{\substack{d_y \in \{k,k+1\} \\ d_z \in \{l,l+1\}}} \mathrm{Fail}(d_y,d_z) \cdot \rho_y(d_y) \cdot \rho_z(d_z) > \sum_{\substack{d_y \in \{k-1,k\} \\ d_z \in \{l+1,l+2\}}} \mathrm{Fail}(d_y,d_z) \cdot \rho'_y(d_y) \cdot \rho'_z(d_z)\ . \tag{2}$$

Note that if $k-l=2$ then the summand regarding $d_y = k$ and $d_z = l+1$ on the left-hand side is the same as the summand regarding $d_y = k-1$ and $d_z = l+2$ on the right-hand side. Hence, to prove (2) it is sufficient to show that

$$\mathrm{Fail}(k,l) > \mathrm{Fail}(k-1, l+1)\ . \tag{3}$$

For this, consider the fixed graph $G_{S-\{y,z\}}$. We classify the right nodes of $G_{S-\{y,z\}}$ according to the following three types:

- We call v *blocked* if v is matched in all matchings of $G_{S-\{y,z\}}$.
- We call v *free* if v is never matched in any matching of $G_{S-\{y,z\}}$.
- We call v *half-free* if v is neither a blocked nor a free node.

Let B be the set of blocked nodes, let F be the set of free nodes, and let HF be the set of half-free nodes. Elements of $\bar{B} = F \cup HF$ are called *non-blocked* nodes. For a moment consider only the non-blocked nodes. For each right node set $V \subseteq \bar{B}$ let H_V be an auxiliary graph with node set V that has an edge between two nodes $v_1, v_2 \in V$ if and only if there exists a matching for $G_{S-\{y,z\}}$ in which v_1 and v_2 simultaneously are *not* matched. Let V be an arbitrary but fixed subset of $\bar{B}$. The following observation is crucial.

Claim 1. If H_V has any edges at all then it is connected.

Proof of Claim. First note that if there is a free node in V then H_V is connected by definition of the edge set of H_V. Therefore it remains to consider the case where all nodes of V are half-free nodes. It is sufficient to show that

if for three nodes v_1, v_2, v_3 from HF the edge (v_1, v_2) is in H_V then one of the edges (v_1, v_3) or (v_2, v_3) must be present as well. Assume for a contradiction (v_1, v_2) is an edge but v_3 is neither adjacent to v_1 nor to v_2. This implies that there are two matchings in $G_{S-\{y,z\}}$, M and M' say, such that in M

- node v_3 is unmatched (v_3 is a non-blocked node), but
- nodes v_1 and v_2 are matched since edges (v_1, v_3) and (v_2, v_3) are not in H_V,

and in M' we have:

- node v_3 is matched (v_3 is a half-free node), but
- v_1 and v_2 are unmatched since edge (v_1, v_2) is in H_V.

Now consider the bipartite multigraph $M \cup M'$ consisting of all edges from both matchings and the corresponding nodes. The graph $M \cup M'$ has the following properties: Nodes on the left side have degree 2 (both matchings are left-perfect). Nodes on the right side have degree 1 or 2, in particular, v_1,v_2,v_3 have degree 1. Hence $M \cup M'$ has only paths and cycles of even length. On all paths and cycles edges from M and M' alternate. Nodes v_1 and v_2 must be at the ends of two distinct paths (since both are incident to M-edges). Node v_3 must be at the end of a path (incident to an M'-edge).

Without loss of generality, we may assume that v_1 and v_3 do not lie on the same path. Starting from M', we get a new matching in which neither v_1 nor v_3 are matched by replacing the M'-edges on the path with v_3 by the M-edges on this path. Therefore there must be an edge (v_1, v_3) in H_V, which contradicts our assumption, proving the claim. □

Now consider the set $\bar{B}$ of non-blocked nodes and the corresponding graph $H_{\bar{B}}$. We define $\sim$ as the following binary relation: $v_1 \sim v_2$, for nodes v_1 and v_2, if (v_1, v_2) is not an edge in $H_{\bar{B}}$.

Claim 2. The relation $\sim$ (no edge) is an equivalence relation.

Proof of Claim. Clearly $\sim$ is reflexive and symmetric. Assume for a contradiction $\sim$ is not transitive. That is, we have three nodes v_1, v_2 and v_3 with $v_1 \sim v_2$ and $v_2 \sim v_3$ but $v_1 \not\sim v_3$. Let $V = \{v_1, v_2, v_3\}$. Since $v_1 \not\sim v_3$, the edge (v_1, v_3) is in $H_{\bar{B}}$ and therefore in H_V. According to Claim 1 H_V must be connected, i.e., H_V and therefore $H_{\bar{B}}$ must contain (v_1, v_2) or (v_2, v_3). Hence $v_1 \not\sim v_2$ or $v_2 \not\sim v_3$, which is a contradiction. □

According to the claim it follows that the right node set T of $G_{S-\{y,z\}}$ can be subdivided into disjoint segments $B \cup I_1 \cup I_2 \cup \ldots = T$, where B is the set of blocked nodes and $I_1, I_2, \ldots$ are the maximal independent sets in $H_{\bar{B}}$ and the equivalence classes of $\sim$, respectively. For each pair I_s, I_t, with $s \neq t$, it holds that $H_{I_s \cup I_t}$ is a complete bipartite graph. Note that each free node leads to a one-element set I_s. With this characterization of $H_{\bar{B}}$ we can express the event that for a fixed neighborhood N_x, $x \in S - \{y, z\}$, which admits a matching for $G_{S-\{y,z\}}$, there is no matching for G_S as follows

$$\{N_y \subseteq B\} \cup \{N_z \subseteq B\} \cup \bigcup_j \{(N_y \cup N_z) \subseteq (B \cup I_j)\} \ . \tag{4}$$

Let $\mathcal{BI}_{S-\{y,z\}}(b, r, i_1, \ldots, i_r)$ be the event that $G_{S-\{y,z\}}$ has $|B| = b$ many blocked nodes and r (nonempty) maximal independent sets according to the definition above, with $|I_j| = i_j$ and $i_1 \leq i_2 \leq \ldots \leq i_r$. Let

$$\begin{aligned}&\mathrm{fail}(d_y, d_z, b, r, i_1, \ldots, i_r) = \\ &\quad \Pr\left(\bar{\mathcal{M}} \mid \mathcal{M}_{S-\{y,z\}}, D_y = d_y, D_z = d_z, \mathcal{BI}_{S-\{y,z\}}(b, r, i_1, \ldots, i_r)\right) \ .\end{aligned}$$

Then (4) implies that

$$\begin{aligned}\mathrm{fail}(d_y, d_z, b, r, i_1, \ldots, i_r) = &\left(\frac{b}{m}\right)^{d_y} + \left(\frac{b}{m}\right)^{d_z} - \left(\frac{b}{m}\right)^{d_y} \cdot \left(\frac{b}{m}\right)^{d_z} \\ &+ \sum_{j=1}^{r} \left[\left(\frac{i_j + b}{m}\right)^{d_y} - \left(\frac{b}{m}\right)^{d_y}\right] \cdot \left[\left(\frac{i_j + b}{m}\right)^{d_z} - \left(\frac{b}{m}\right)^{d_z}\right] \ .\end{aligned}$$

Using the law of total probability we can rewrite the value $\mathrm{Fail}(d_y, d_z)$ (line below (1)) as follows:

$$\begin{aligned}\mathrm{Fail}(d_y, d_z) = \sum_{(b,r,i_1,\ldots,i_r)} &\mathrm{fail}(d_y, d_z, b, r, i_1, \ldots, i_r) \\ &\cdot \Pr(\mathcal{BI}_{S-\{y,z\}}(b, r, i_1, \ldots, i_r) \mid \mathcal{M}_{S-\{y,z\}}) \ .\end{aligned}$$

We will abbreviate $\mathrm{fail}(d_y, d_z, b, r, i_1, \ldots, i_r)$ by $\mathrm{fail}(d_y, d_z)$ for the rest of the paper. In order to prove (3) it is sufficient to show

$$\mathrm{fail}(k, l) > \mathrm{fail}(k-1, l+1) \ , \tag{5}$$

for each $\mathcal{BI}$-vector $(b, r, i_1, \ldots, i_r)$. Let $\gamma_j = i_j/m$ and let $\beta = b/m$. Thus,

$$\mathrm{fail}(k, l) = \beta^k + \beta^l - \beta^{k+l} + \sum_{j=1}^{r} \left[(\gamma_j + \beta)^k - \beta^k\right] \cdot \left[(\gamma_j + \beta)^l - \beta^l\right] \ . \tag{6}$$

Hence, inequality (5) holds if and only if

$$\begin{aligned}\beta^k + \beta^l - \beta^{k-1} - \beta^{l+1} > &\sum_{j=1}^{r} \left[(\gamma_j + \beta)^{k-1} - \beta^{k-1}\right] \cdot \left[(\gamma_j + \beta)^{l+1} - \beta^{l+1}\right] \\ &- \left[(\gamma_j + \beta)^k - \beta^k\right] \cdot \left[(\gamma_j + \beta)^l - \beta^l\right] \\ \Leftrightarrow \quad (1-\beta) \cdot (\beta^l - \beta^{k-1}) > &\sum_{j=1}^{r} \gamma_j \cdot \underbrace{\left[\beta^l \cdot (\gamma_j + \beta)^{k-1} - \beta^{k-1} \cdot (\gamma_j + \beta)^l\right]}_{\phi(l,k,\gamma_j,\beta)} \ .\end{aligned} \tag{7}$$

Note that if $r = 1$ there is no matching for G_S. Hence we are only interested in the case $r \geq 2$, which implies that $i_j < m - b$ and $\gamma_j < 1 - \beta$, respectively.

Consider the right-hand side of the inequality. The expression within the square brackets increases monotonically with increasing γ_j, since we have

$$\frac{\partial\phi(l,k,\gamma_j,\beta)}{\partial\gamma_j} = (k-1)\cdot\beta^l\cdot(\gamma_j+\beta)^{k-2} - l\cdot\beta^{k-1}\cdot(\gamma_j+\beta)^{l-1} \stackrel{!}{>} 0$$
$$\Leftrightarrow \frac{k-1}{l}\cdot(\gamma_j+\beta)^{k-l-1} > \beta^{k-l-1} ,$$

and the last inequality holds because of $k-l \geq 2$ and $\gamma_j+\beta > \beta$. Therefore replacing γ_j with $1-\beta$ within ϕ and using that $\sum_{j=1}^{r}\gamma_j = 1-\beta$ strictly increases the right-hand side of (7) and yields the left-hand side of (7). But since we assume $\gamma_j < 1-\beta$ the strict inequality holds. Due to the fact that the event $\{r \geq 2\}$ has positive probability Lemma 2 follows. ■

2.2 Optimal Distributions Use Only Two Neighboring Degrees

In this section we prove Lemma 3. Consider the probability mass functions ρ_y and ρ_z for the degrees D_y and D_z respectively. Let $\lfloor\Delta_y\rfloor = l$ and $\lfloor\Delta_z\rfloor = l-1$ as well as $p = 1-(\Delta_y - \lfloor\Delta_y\rfloor)$ and $q = 1-(\Delta_z - \lfloor\Delta_z\rfloor)$. By the hypothesis of the lemma we have

$$\rho_y(l) = p,\ \rho_y(l+1) = 1-p \qquad \rho_z(l-1) = q,\ \rho_z(l) = 1-q ,$$

with $p \in (0,1)$ and $q \in (0,1)$. To prove Lemma 3 we will show that changing ρ_y to ρ'_y and ρ_z to ρ'_z, via

$$\rho'_y(l) = p+\varepsilon,\ \rho'_y(l+1) = 1-p-\varepsilon \quad \rho'_z(l-1) = q-\varepsilon,\ \rho'_z(l) = 1-q+\varepsilon ,$$

for some small perturbation $\varepsilon \neq 0$ will strictly increase the probability that G_S has a matching, while it does not change $\bar{\Delta}$. Hence as in the proof of Lemma 2 we will show that

$$\Pr\left(\bar{\mathcal{M}}_S \mid \rho_y, \rho_z\right) > \Pr\left(\bar{\mathcal{M}}_S \mid \rho'_y, \rho'_z\right) .$$

As before we fix the neighborhood N_x for the remaining elements $x \in S-\{y,z\}$ and therefore the graph $G_{S-\{y,z\}}$. As in Lemma 2 we conclude that it is sufficient to show that for some perturbation term $\varepsilon \neq 0$ we have

$$\sum_{\substack{d_y\in\{l,l+1\}\\ d_z\in\{l-1,l\}}} \mathrm{Fail}(d_y,d_z)\cdot\rho_y(d_y)\cdot\rho_z(d_z) > \sum_{\substack{d_y\in\{l,l+1\}\\ d_z\in\{l-1,l\}}} \mathrm{Fail}(d_y,d_z)\cdot\rho'_y(d_y)\cdot\rho'_z(d_z) .$$

Subtracting the left-hand side from right-hand side gives

$$\left[-\varepsilon^2-\varepsilon\cdot(p-q)\right]\cdot\underbrace{\left[\mathrm{Fail}(l,l-1)+\mathrm{Fail}(l+1,l)-\mathrm{Fail}(l,l)-\mathrm{Fail}(l+1,l-1)\right]}_{K_0}$$
$$-\varepsilon\cdot\underbrace{\left[\mathrm{Fail}(l+1,l-1)-\mathrm{Fail}(l,l)\right]}_{K_1} < 0$$
$$\Leftrightarrow \quad -\varepsilon^2\cdot K_0 - \varepsilon\cdot\underbrace{\left[(p-q)\cdot K_0 + K_1\right]}_{L} < 0 . \tag{8}$$

From (3), which was proven in Lemma 2, it follows that $K_1 > 0$. There are three cases.

$K_0 = 0$. Since we have $K_1 > 0$, it is easy to see that (8) holds for $\varepsilon > 0$.

$K_0 > 0$. Regardless whether L is zero, positive, or negative, (8) holds for some small $\varepsilon \neq 0$.

$K_0 < 0$. The only critical case would be $L = 0$, but we will show that it holds $K_1 > -K_0$ and therefore $L > 0$, implying that (8) holds for small $\varepsilon > 0$.

Inequality $K_1 > -K_0$ holds if and only if

$$\text{Fail}(l+1,l) + \text{Fail}(l,l-1) > 2 \cdot \text{Fail}(l,l) \ .$$

As before we will simply show the sufficient condition

$$\text{fail}(l+1,l) + \text{fail}(l,l-1) > 2 \cdot \text{fail}(l,l) \ .$$

Using (6) in combination with the substitutions $\gamma_j = i_j/m$ and $\beta = b/m$ the condition can be written as

$$\begin{aligned}(1-\beta)^2 \cdot \left[\beta^{l-1} - \beta^{2l-1}\right] > &\sum_{j=1}^{r} (1-\beta)^2 \cdot \left[(\gamma_j+\beta)^l \cdot \beta^{l-1} - \beta^{2l-1}\right] \\ &- \sum_{j=1}^{r} [1-(\gamma_j+\beta)]^2 \cdot \left[(\gamma_j+\beta)^{2l-1} - (\gamma_j+\beta)^{l-1} \cdot \beta^l\right] .\end{aligned}$$

Note that the subtrahend of the right-hand side is non negative. Hence it is sufficient to show that

$$(1-\beta)^2 \cdot \left[\beta^{l-1} - \beta^{2l-1}\right] > (1-\beta)^2 \cdot \sum_{j=1}^{r} (\gamma_j+\beta)^l \cdot \beta^{l-1} - r \cdot (1-\beta)^2 \cdot \beta^{2l-1} \ . \quad (9)$$

Bounding $\sum_{j=1}^{r} (\gamma_j+\beta)^l$ using the binomial theorem gives

$$\begin{aligned}\sum_{j=1}^{r} (\gamma_j+\beta)^l &= \sum_{j=1}^{r} \sum_{i=0}^{l} \binom{l}{i} \cdot \gamma_j^i \cdot \beta^{l-i} = r \cdot \beta^l + \sum_{i=1}^{l} \binom{l}{i} \cdot \beta^{l-i} \cdot \sum_{j=1}^{r} \gamma_j^i \\ &< r \cdot \beta^l + \sum_{i=1}^{l} \binom{l}{i} \cdot \beta^{l-i} \cdot \left[\sum_{j=1}^{r} \gamma_j\right]^i = (r-1) \cdot \beta^l + 1 \ ,\end{aligned}$$

where the last step follows from $\sum_{j=1}^{r} \gamma_j = 1 - \beta$. Substituting $\sum_{j=1}^{r} (\gamma_j+\beta)^l$ with $(r-1) \cdot \beta^l + 1$ shows that (9) holds and thus the lemma. ■

3 A Conjecture: Essentially Two Different Strategies

In this section, we give evidence for Conjecture 1, which says that essentially two types of distributions may be optimal: one in which all keys are given fixed

degrees l or $l+1$, and one in which each node chooses one of l and $l+1$ at random, governed by the same distribution on $\{l, l+1\}$. We indicate under what circumstances the one or the other is best.

Assume we are in the situation of Theorem 1 (*ii*), i.e., $l < \bar{\Delta} < l+1$ for some integer constant $l \geq 2$ and it holds $\rho_x(l) \in [0,1]$ and $\rho_x(l+1) = 1 - \rho_x(l)$, for each x from S. Let y and z be two arbitrary but fixed elements of S with

$$\rho_y(l) = p,\ \rho_y(l+1) = 1-p \qquad \rho_z(l) = q,\ \rho_z(l+1) = 1-q\ ,$$

for $p \in [0,1]$ and $q \in [0,1]$. We would like to know if the matching probability increases if we change the probability mass functions ρ_y and ρ_z to ρ'_y and ρ'_z, via

$$\rho'_y(l) = p+\varepsilon,\ \rho'_y(l+1) = 1-p-\varepsilon \quad \rho'_z(l) = q-\varepsilon,\ \rho'_z(l+1) = 1-q+\varepsilon\ ,$$

for some $\varepsilon > 0$. We note the following.

1. If $p \geq q$, i.e., $\Delta_y \leq \Delta_z$, this modification would move both means towards the boundary of the interval $[l, l+1]$. Moving a mean beyond the boundary cannot increase the matching probability since this would be a contradiction to Lemma 3.
2. If $p < q$, i.e., $\Delta_y > \Delta_z$, this modification would move both means towards each other.

As in Lemma 3 it can be shown that the matching probability increases iff

$$\sum_{d_y, d_z \in \{l, l+1\}} \mathrm{Fail}(d_y, d_z) \cdot \rho_y(d_y) \cdot \rho_z(d_z) > \sum_{d_y, d_z \in \{l, l+1\}} \mathrm{Fail}(d_y, d_z) \cdot \rho'_y(d_y) \cdot \rho'_z(d_z)\ .$$

This inequality is equivalent to

$$[-\varepsilon^2 - \varepsilon \cdot (p-q)] \cdot \underbrace{[\mathrm{Fail}(l,l) - 2 \cdot \mathrm{Fail}(l, l+1) + \mathrm{Fail}(l+1, l+1)]}_{K} < 0\ , \quad (10)$$

utilizing the symmetry $\mathrm{Fail}(l+1, l) = \mathrm{Fail}(l, l+1)$. Whether inequality (10) holds or not depends on K, which is independent of y, z and p, q. There are three cases.

$K_0 = 0$. The modifications to ρ_y and ρ_z do not change the failure probability. This case seems unlikely since there would be an infinite number of optimal sequences of probability mass functions; hence we will ignore this case for the rest of the discussion.

$K > 0$. Arrange that $p \geq q$ (if necessary interchange y and z). Then increasing p and decreasing q (moving the means away from each other) increases the success probability.

$K < 0$. Arrange that $p < q$ (if $p = q$ do nothing). Again, increasing p and decreasing q (moving the means closer together) increases the success probability.

Using the same method as in Lemmas 2 and 3 it is not possible to show that always $K < 0$ or always $K > 0$ happens. To see this, we try to show $K > 0$ which is equivalent to proving that

$$\mathrm{Fail}(l,l) + \mathrm{Fail}(l+1, l+1) > 2 \cdot \mathrm{Fail}(l, l+1)\ . \quad (11)$$

As before we only consider the sufficient condition

$$\text{fail}(l,l) + \text{fail}(l+1,l+1) > 2 \cdot \text{fail}(l,l+1) \ . \tag{12}$$

This inequality is equivalent to

$$\begin{aligned} & 2 \cdot \beta^l - \beta^{2l} + \sum_{j=1}^{r} [(\gamma_j + \beta)^l - \beta^l]^2 + 2 \cdot \beta^{l+1} - \beta^{2l+2} + \sum_{j=1}^{r} [(\gamma_j + \beta)^{l+1} - \beta^{l+1}]^2 \\ > \quad & 2 \cdot \beta^l + 2 \cdot \beta^{l+1} - 2 \cdot \beta^{2l+1} + 2 \cdot \sum_{j=1}^{r} [(\gamma_j + \beta)^l - \beta^l] \cdot [(\gamma_j + \beta)^{l+1} - \beta^{l+1}] \ , \end{aligned}$$

where we use the substitutions $\gamma_j = i_j/m$ and $\beta = b/m$. Moving the $\sum_j$-terms to the left and the remaining β-terms to the right gives

$$\sum_{j=1}^{r} \left[(\gamma_j + \beta)^l \cdot (1 - \gamma_j - \beta) - \beta^l \cdot (1 - \beta)\right]^2 > \beta^{2l} \cdot (1 - \beta)^2 \ .$$

However, this inequality may hold or may not hold depending on γ_j and β. For example, consider the events

1. $\{\beta = 0\}$, then the inequality is true for all $r \geq 2$, and
2. $\{r = 2, \gamma_1, \gamma_2 = 1/(2 \cdot l), \beta = 1 - 1/l\}$, then the inequality is false.

Note that events 1. and 2. have positive probability.

It follows that there exists graphs $G_{S-\{y,z\}}$ in which (12) is true as well as graphs in which (12) is false. Hence, it could be possible that there are nodes y_1, z_1 with $K < 0$ (their means should be made equal), and nodes y_2, z_2 with $K > 0$ (their means should be moved away from each other). So hypothetically, it could be optimal when S is subdivided into 3 disjoint sets S_l,S_{l+1}, and $S_{l,l+1}$ where each node from S_l has fixed degree l and each node from S_{l+1} has fixed degree $l+1$ and each node from $S_{l,l+1}$ has the same mean $\Delta \in (l, l+1)$, and the degree of each node is concentrated on l and $l+1$. But this would mean if we assume such an "optimal situation" and we have three different nodes, say y_1, y_2 and z, where $y_1, y_2 \in S_{l,l+1}$ and $z \in S_l$, then it must hold $K > 0$ for $G_{S-\{y_1,z\}}$ and $K < 0$ for $G_{S-\{y_1,y_2\}}$ which seems unlikely since $S - \{y_1, z\}$ and $S - \{y_1, y_2\}$ differ in only one node. (Recall that $K = 0$ does not seem plausible, either.) Therefore we conjecture that it is optimal if it holds

1. either $S = S_l \cup S_{l+1}$, that is for each x from S the mean Δ_x is fixed to one of the interval borders l and $l+1$, and therefore a fixed fraction of $\lceil \bar{\Delta} \rceil - \bar{\Delta}$ of the nodes have degree l (assuming that $\bar{\Delta} \cdot n$ is an integer),
2. or $S = S_{l,l+1}$, that is it holds $\bar{\Delta} = \Delta_x$ for each x from S, and therefore the number of nodes of degree l follow a binomial distribution with parameters n and $\lceil \bar{\Delta} \rceil - \bar{\Delta}$.

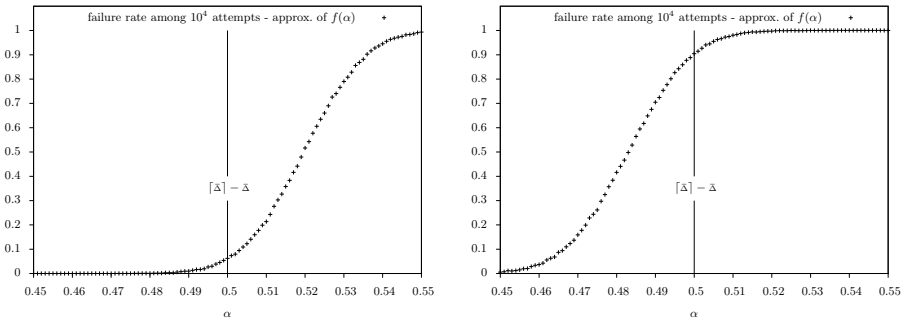
For the rest of the discussion we only focus on these two degree distributions (fixed and binomial) and we try to argue under which conditions case 1 is optimal and when case 2 is optimal.

Again our starting point is (11) and $K > 0$ respectively. Now fix the degree of all left nodes from G_S and let α be the fraction of nodes from S with degree l as well as let let α' be the fraction of nodes from $S - \{y, z\}$ with degree l. Then there are three situations to distinguish according to the degrees of y and z.

(*i*) $\alpha = \alpha' + 2/n$, that is y and z have degree l,
(*ii*) $\alpha = \alpha' + 1/n$, that is one node has degree l the other node has degree $l+1$,
(*iii*) $\alpha = \alpha'$, that is both nodes have degree $l+1$.

Inequality (11) states that the increase of the failure probability from (*ii*) to (*i*) is larger than the increase of the failure probability from (*iii*) to (*ii*) for all α' from $[0, 1]$, that is, the failure probability as a function of α should be convex (while strictly monotonically increasing). Experimental results as shown in Figure 1 suggest that this is not the case in general. In fact there are two different situations for fixed $\bar{\Delta}$ shown in Figures 1(a) and 1(b). Let $f(\alpha)$ denote the failure probability as a function of α. If $c < c^*(\bar{\Delta})$ then f is convex in a neighborhood of $\lceil \bar{\Delta} \rceil - \bar{\Delta}$. Using Jensen's inequality it follows that the failure probability for fixed degree distribution $f(\lceil \bar{\Delta} \rceil - \bar{\Delta})$ is smaller than the failure probability according to the binomial distribution $\sum_{i=0}^{n} f(i/n) \cdot \binom{n}{i} \cdot (\lceil \bar{\Delta} \rceil - \bar{\Delta})^i \cdot (1 - \lceil \bar{\Delta} \rceil + \bar{\Delta})^{n-i}$, ignoring the right tail of the binomial distribution that reaches the concave part of $f(\alpha)$, since the tail covers only an exponentially small probability mass. If $c > c^*(\bar{\Delta})$ then f is concave in a neighborhood of $\lceil \bar{\Delta} \rceil - \bar{\Delta}$ and the binomial degree distribution leads to a smaller failure probability than the fixed degree distribution.

In order to back up this observation, an additional experiment was done which directly compares the failure rates for degree distributions around the threshold. The results are shown in Figure 2. They confirm the conjecture that if $c < c^*(\bar{\Delta})$ then the fixed degree distribution is optimal, and if $c > c^*(\bar{\Delta})$ then the binomial degree distribution is optimal.



(a) $c < c^*(\bar{\Delta})$, $c = 0.956$ (b) $c > c^*(\bar{\Delta})$, $c = 0.958$

Fig. 1. Rate of random bipartite graphs with $D_x \in \{3, 4\}$, $\bar{\Delta} = 3.5$, $m = 10^5$, that have no matching, as a function of α (the fraction of left nodes with degree 3). The plots show that the failure function $f(\alpha)$ has probably a transition from convex to concave. The theoretical threshold $c^*(3.5)$ is approximately 0.957.

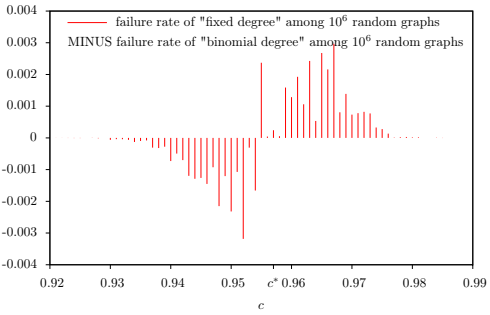


Fig. 2. Difference of the failure rate of graphs $G\big(\bar{\Delta}, (\rho_x)_{x\in S}\big)$ with $D_x \in \{3, 4\}$, $\bar{\Delta} = 3.5, m = 10^3$ and different $(\rho_x)_{x\in S}$, as a function of $c = n/m$. Minuend is the failure rate using fixed degree, that is $\rho_x(3) \in \{0, 1\}$, for each $x \in S$. Subtrahend is the failure rate using binomial degree distribution that is $\rho_x(3) = 0.5$, for each $x \in S$.

4 Conclusion

We found (near) optimal degree distributions for matchings in bipartite multigraphs where each left node chooses its right neighbors randomly with repetition according to its assigned degree. For the case that the number of left nodes is linear in the number of right nodes we showed that these distributions give matching thresholds that are the same as the known thresholds for regular/irregular k-ary cuckoo hashing; and in the case of near optimal degree distributions we conjectured the optimal distribution as a function of the rate of left and right nodes.

Acknowledgment. The authors would like to thank an anonymous reviewer for pointing out a gap in an earlier version of the proof of Lemma 3.

References

1. Dietzfelbinger, M., Goerdt, A., Mitzenmacher, M., Montanari, A., Pagh, R., Rink, M.: Tight Thresholds for Cuckoo Hashing via XORSAT. CoRR arXiv:0912.0287 (2009)
2. Dietzfelbinger, M., Goerdt, A., Mitzenmacher, M., Montanari, A., Pagh, R., Rink, M.: Tight Thresholds for Cuckoo Hashing via XORSAT. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part I. LNCS, vol. 6198, pp. 213–225. Springer, Heidelberg (2010)
3. Dietzfelbinger, M., Rink, M.: Towards Optimal Degree-distributions for Left-perfect Matchings in Random Bipartite Graphs. CoRR arXiv:1203.1506 (2012)
4. Fountoulakis, N., Panagiotou, K.: Orientability of Random Hypergraphs and the Power of Multiple Choices. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part I. LNCS, vol. 6198, pp. 348–359. Springer, Heidelberg (2010)
5. Frieze, A.M., Melsted, P.: Maximum Matchings in Random Bipartite Graphs and the Space Utilization of Cuckoo Hashtables. CoRR arXiv:0910.5535 (2009)
6. Luby, M.G., Mitzenmacher, M., Shokrollahi, M.A., Spielman, D.A.: Efficient erasure correcting codes. IEEE Transactions on Information Theory 47(2), 569–584 (2001)
7. Rink, M.: On Thresholds for the Appearance of 2-cores in Mixed Hypergraphs (in preparation)

Robust Sensor Range for Constructing Strongly Connected Spanning Digraphs in UDGs

Stefan Dobrev[1,⋆], Evangelos Kranakis[2,⋆⋆], Oscar Morales Ponce[2,⋆⋆⋆], and Milan Plžík[3,†]

[1] Institute of Mathematics, Slovak Academy of Sciences, Bratislava, Slovak Republic
Stefan.Dobrev@savba.sk
[2] School of Computer Science, Carleton University, Ottawa, Canada
kranakis@scs.carleton.ca, omponce@connect.carleton.ca
[3] Comenius University, Bratislava, Slovak Republic
milan.plzik@gmail.com

Abstract. We study the following problem: Given a set of points in the plane and a positive integer $k > 0$, construct a geometric strongly connected spanning digraph of out-degree at most k and whose longest edge length is the shortest possible. The motivation comes from the problem of replacing omnidirectional antennae in a sensor network with k directional antennae per sensor so that the resulting sensor network is strongly connected. The contribution of this is paper is twofold:

1) We introduce a notion of *robustness* of the radius in geometric graphs. This allows us to provide stronger lower bounds for the edge length needed to solve our problem, while nicely connecting two previously unrelated research directions (graph toughness and multiple directional antennae).

2) We present novel upper bound techniques which, in combination with stronger lower bounds, allow us to improve the previous approximation results for the edge length needed to achieve strong connectivity for $k = 4$ (from $2\sin(\pi/5)$ to optimal) and $k = 3$ (from $2\sin(\frac{\pi}{4})$ to $2\sin(\frac{2\pi}{9})$).

1 Introduction

Consider a set of sensors in the plane such that each sensor has k directional antennae and the sum of angles covered by these antennae is at most ϕ. What is the smallest communication range r such that there exists an orientation of the antennae such that the resulting communication graph is strongly connected? When $\phi = 0$ the problem is equivalent to the problem of finding a spanning digraph of maximum out-degree k that is strongly connected and minimizes the maximal edge length used.

⋆ Supported by VEGA 2/0136/12.
⋆⋆ Supported in part by NSERC and MITACS grants.
⋆⋆⋆ Supported by MITACS Postdoctoral Fellowship.
† Supported by VEGA 2/0136/12.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 112–124, 2012.

Before giving the formal definition of the problem we will introduce the following notation. Let $\overrightarrow{G}$ denote a weighted directed geometric graph and let $\Delta^+(\overrightarrow{G})$ denote the maximum out-degree of $\overrightarrow{G}$. For any edge (u, v) let $w(u, v)$ denote its weight. We refer to our problem as the *Bottleneck Strongly Connected Spanning Digraph with Bounded Out-Degree* (or BSCBOD for short) and define it formally as follows:

Problem 1. Given a weighted complete digraph $\overrightarrow{G}$ and an integer $k \geq 1$, determine a strongly connected spanning subgraph $\overrightarrow{H}$ such that $\Delta^+(\overrightarrow{H}) \leq k$ and the maximum weight is minimum, i.e.,

$$\min\{\max_{(u,v)\in\overrightarrow{H}} w(u, v) : \overrightarrow{H} \text{ is a strongly connected and } \Delta^+(\overrightarrow{H}) \leq k\}$$

Recall that the UDG (Unit Disk Graph) of a set of n points P with parameter r is the geometric graph, denoted by $U(P; r)$, where the set of vertices is P and vertices u, v are adjacent (with a straight line segment) if and only if $d(u, v) \leq r$, where $d(\cdot, \cdot)$ denotes the Euclidean distance between u and v.

Now we can rephrase our problem in the geometric setting as follows:

Problem 2. Given a set of points P and an integer $k \geq 1$, find the smallest edge length r such that UDG $U(P; r)$ has a strongly connected spanning directed subgraph $\overrightarrow{H}$ with $\Delta^+(\overrightarrow{H}) \leq k$ and construct $\overrightarrow{H}$.

1.1 Related Work

When $k = 1$ the problem is equivalent to the well-studied problem of bottleneck traveling salesman problem. Parker and Radin [8] give a 2-approximation when the weights satisfy the triangle inequality and show that it is not possible to approximate to $2 - \epsilon$ unless $P = NP$.

In the Euclidean version of the problem, weights are defined by the Euclidean distance between the two points in $\mathbf{R}^2$. Papadimitriou [7] shows that in this setting the problem remains NP-Complete.

In the context of sensor networks Caragiannis et al. [2] proposed the problem of replacing omnidirectional antennae with directional antennae. They study the setting when each sensor has one directional antenna of a given angle. They showed that the problem is NP-hard when the angle is less than $2\pi/3$ and the communication range is less than $\sqrt{3}$ times the maximum length of an edge of the Euclidean Minimum Spanning Tree (MST) (denoted by r_{MST}).

In [3], the authors study the problem in the context of multiple directional antennae in a wireless sensor network. In this setting, sensors are equipped with directional antennae of given beam-width (angle) and range (radius); their goal is to give algorithms for orienting the antennae and study angle/range trade-offs for achieving strong connectivity. On the one hand, when $k \geq 5$, the problem is trivially solved by using each edge of the MST in both directions, since there always exists a MST (Minimum Spanning Tree) on a set of points with maximum degree five [6]. On the other hand, they show that the problem is NP-complete

when $k = 2$ even for a scaling factor of 1.3 and/or sum of angles less than $9\pi/20$. They also give an algorithm to compute upper bounds for the cases when $k = 2, 3, 4$. Table 1 summarizes the results given in [3] relating to our problem. A comprehensive survey is presented in [5].

Table 1. Results given in [3]

Out-degree	Lower Bound	Upper Bound	Approx. Ratio	Complexity
4	r_{MST}	$2\sin(\pi/5)r_{MST}$	$2\sin(\pi/5)$	$O(n \log n)$
3	r_{MST}	$2\sin(\pi/4)r_{MST}$	$\sqrt{2}$	$O(n \log n)$
2	r_{MST}	$2\sin(\pi/3)r_{MST}$	$\sqrt{3}$	$O(n \log n)$
2	-	-	≤ 1.3	NP-Complete

Another similar but less related problem is the minimum spanning tree with degree at most k. In [4] Francke and Hoffmann show that it is NP-Hard to decide whether a given set S of n points in the plane admits a spanning tree of maximum degree four whose sum of edge lengths does not exceed a given threshold k.

1.2 Results and Outline of the Paper

This paper contains three major contributions, presented in sections 2, 3, and 4, respectively. In Section 2 we introduce the new concepts of strong t-robustness (σ_t) and weak t-robustness (α_t) of a UDG radius, which are closely related to the well studied concept of graph toughness. The primary motivation comes from the observation that the strong $1/t$-robustness and weak $1/t$-robustness provide a more refined and higher lower bound than the weight of the longest edge of the MST for the problem we study[1].

In Section 3 we present our optimal algorithm for $k = 4$. In Section 4 we combine the technique used in Section 3 and a modified version of the technique from [3]. This allows us to save an out-going arc thus improving the approximation ratio in [3] to $2\sin(2\pi/9)$. Due to space constraints technical proofs of Section 2 and algorithms of Section 3 and 4 appear in the full paper.

Table 2 summarizes the main results of Section 4. We conclude in Section 5 with a discussion of various related issues and open problems.

Table 2. Summary of results obtained in this paper

Out-degree	Lower Bound	Upper Bound	Approx. Ratio	Complexity
4	$\sigma_{1/4}$	$\alpha_{1/4}$	1	$O(n \log n)$
3	$\sigma_{1/3}$	$2\sin(2\pi/9)\alpha_{1/3}$	$\leq 2\sin(2\pi/9)$	$O(n \log n)$

[1] While the strong robustness provides stronger lower bound, the weak robustness is easier to compute and use.

2 Robustness of Unit Disk Graphs

The concept of toughness of a graph as a measure of graph connectivity has been extensively studied in the literature (see the survey [1]). Intuitively, graph toughness measures the resilience of the graph to fragmentation after subgraph removal. Given a graph G, let us denote by $\omega(G)$ the number of components of G.

As defined in [1], a graph G is t-tough if $|S| \geq t\omega(G \setminus S)$, for every subset S of the vertex set of G with $\omega(G \setminus S) > 1$. The toughness of G, denoted $\tau(G)$, is the maximum value of t for which G is t-tough (taking $\tau(K_n) = \infty$, for all $n \geq 1$).

We are interested in the toughness of UDGs over a given point set P, and in particular how does the toughness of $U(P; r)$ depend on the antenna radius r. This is expressed in the following definitions:

Definition 1 (Strong and Weak t-robustness for UDG radius). *Let P be a set of points in the plane.*

1. *A subset $S \subseteq P$ is called t-tough if $\omega(U(P \setminus S; r)) \leq |S|/t$. Similarly, a point u is called t-tough if the singleton $\{u\}$ is t-tough.*
2. *The strong t-robustness of the set of points P, denoted by $\sigma_t(P)$, is the infimum taken over all radii $r > 0$ such that for all $S \subseteq P$, the set S is t-tough for the radius r.*
3. *The weak t-robustness of the set of points P, denoted by $\alpha_t(P)$, is the infimum taken over all radii $r > 0$ such that for all $u \in P$, the point u is t-tough for the radius r.*

Note that when P is finite (as in the rest of this paper), it is sufficient to consider only the radii corresponding to pairwise distances between the points of P, in which case it is sufficient to consider minimum instead of infimum.

In order to solve BSCBOD, we are interested in the optimal radius that allows achieving strong connectivity for a given maximum out-degree k.

Definition 2 (Optimal Radius). *For $k \geq 1$, define $r_k(P)$ to be the minimum radius necessary to construct a strongly connected spanning digraph $\overrightarrow{H}$ such that $\Delta^+(\overrightarrow{H}) \leq k$.*

The following theorem motivates our study of robustness of UDG radius:

Theorem 1. $\sigma_{1/k}(P) \leq r_k(P)$

Proof. By contradiction. Assume there exist P and k such that $r_k(P) < \sigma_{1/k}(P)$. Therefore, there must exist $S \subseteq P$ such that S is not $1/k$-tough for radius $r_k(P)$. From Definition 1 we have $\omega(U(P \setminus S, r_k(P))) > k|S|$, i.e. removing S creates more than $k|S|$ connected components. This is in contradiction with the fact that there exists a solution for BSCBOD with radius $r_k(P)$ and maximal out-degree k, as there are not enough antennae in vertices of S to reach all components of $U(P \setminus S, r_k(P))$. □

Let $r_{MST}(P)$ denote the length of the longest edge of the MST of P. From the minimality of the MST, $r_k(P)$ cannot be less than $r_{MST}(P)$. Therefore, Theorem 1 will yield stronger lower bounds than $r_{MST}(P)$ when $k < 5$.

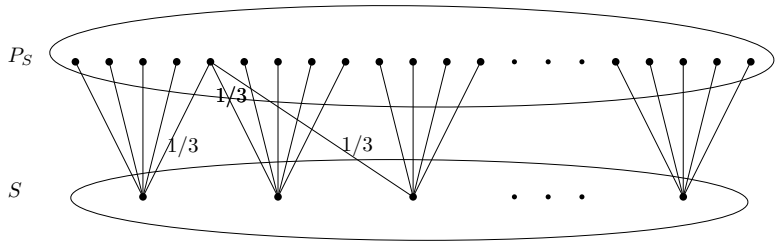


Fig. 1. Bipartite graph $H_S(G) = (S \cup P_S, E_S)$

2.1 Efficiently Computing Weak t-Robustness

The weak t-robustness $\alpha_t(P)$ of a set P of points refers to single points and as such it can be computed in polynomial time for a set of n points. A naive algorithm takes $O(n^4)$ time: test in $O(n^2)$ time for a given radius, checking each of the $O(n^2)$ possible radii given by distances between pairs of points. However, this is not the case for the strong t-robustness $\sigma_t(P)$ which depends on all subsets $S \subseteq P$.

As each singleton vertex is also a subset of P, Definition 1 directly yields

Lemma 1. *$\alpha_t(P) \leq \sigma_t(P)$, for all t.*

It is not difficult to see that any connected UDG is at least 1/5-robust since the removal of any vertex leaves a maximum of five connected components.

The first important observation is that the weak 1/4-robustness and strong 1/4-robustness of a set of points coincide and as a consequence the 1/4-strong-robustness can be computed efficiently.

Theorem 2. *For any set P of points, $\alpha_{1/4}(P) = \sigma_{1/4}(P)$.*

Proof. In view of the observation in Lemma 1 above we only need to show that $\alpha_{1/4}(P) \geq \sigma_{1/4}(P)$. We will show that if for some r the graph $G = U(P; r)$ is not 1/4-robust then there exists a vertex v such that $U(P \setminus \{v\}; r)$ has 5 connected components. Let S be the set such that $U(P \setminus S; r)$ has at least $4|S|+1$ connected components. Consider the bipartite graph $H_S(G) = (S \cup P_S, E_S)$ defined as follows: P_S is the set of connected components of $G \setminus S$, $E_S = \{\{u, v\} : u \in S, v \in P_S$ such that there is an edge in G between u and a vertex from $P_S\}$.

Note that the maximal degree of vertices from S is 5: Assume the converse, i.e. there is a vertex $u \in S$ of degree at least 6. This means there exist edges $\{u, v_i\}$ for $i = 1, 2, \ldots 6$ such that $\{u, v_i\} \in G$ and each v_i is from a different connected component of $U(P \setminus S; r)$. However, at least one of the angles between these six edges must be at most $\pi/3$ and therefore they cannot all lead to different connected components.

Let us assign a weight $w(e)$ to each edge as follows: Each vertex v from P_S equally distributes weight 1 among its incident edges, i.e., $1/deg(v)$. Since P_S is an independent set of $H_S(G)$, each edge is given a unique weight $1/i$ for some i.

Note that

$$\sum_{u \in S} \sum_{e=\{u,\cdot\}} w(e) = \sum_{v \in P_S} \sum_{e=\{\cdot,v\}} w(e) = \sum_{v \in P_S} 1 = |P_S| > 4|S|.$$

Therefore, there must exist a vertex u from S such that $\sum_{e=\{u,\cdot\}} w(e) > 4$. However, since the weight of each edge is $1/i$ for some i and the maximal degree of vertices in S is at most 5, this is only possible if at least four edges incident to u have weight 1 and one has weight greater than 0, i.e., $U(P \setminus \{v\}; r)$ has 5 connected components. □

In fact, weak 1/4-robustness (and more generally, weak $1/i$-robustness for $i < 5$) can be computed much more efficiently than a trivial $O(n^4)$ algorithm. The basic idea is to maintain a tree H of red and black vertices, where the black vertices represent bi-connected components (blocks) and the red vertices represent the separator vertices. Each vertex v of P points to (in variable $h(v)$) its representative in H. Furthermore, each red vertex maintains in variable $c(v)$ (separator degree) the number of components it connects. Initially, H starts as the MST of P; subsequently the edges are added according to their increasing length. The $h(\cdot)$ pointers allow to efficiently determine whether an edge connects two different vertices of H. If adding an edge e closes a cycle in H, the separator degrees of the red vertices (except those incident to e) in this cycle are reduced by one. The process is repeated by taking progressively longer edges until no red vertices remain. As the $h(\cdot)$ pointers can be maintained using standard techniques with the overall cost of $O(n \log n)$ and the overall cost of processing and collapsing created cycles is $O(n)$, the overall cost if $O(|E| + n \log n)$, where E is the set of processed edges.

The second idea of the algorithm comes from the observation that only a set of edges of size $O(n)$ will ever be processed.

Theorem 3. *For every point of P, weak $1/i$-robustness, for $1 \leq i < 5$, can be computed in time $O(|P| \log |P|)$.*

The proof of the theorem will be provided after giving the following lemmas. Let us denote by $TUDG(P, r)$ the graph $T \cup UDG(P, r)$, where T is the MST of P.

Lemma 2. *The following invariant holds for $\mathcal{A}\alpha_{1/i}$ at the beginning of each iteration of the loop. Let $e = \{u, v\}$ be the last processed edge.*

- *H is a tree*
- *A vertex is red if and only if it is a separator vertex of $TUDG(P, d(u, v))$. Furthermore, for every red vertex $w : h(w) = w$ and $c(w)$ is the number of connected components of $TUDG(P, d(u, v)) \setminus \{w\}$.*
- *$\forall x, y \in P : h(x) = h(y)$ if and only if x and y lie in the same block of $TUDG(P, d(u, v))$.*

Proof. By induction over loop iterations. The base step at the beginning of the first iteration of the loop follows from the construction and the definitions.

The induction step:

Algorithm 1. Algorithm $\mathcal{A}\alpha_{1/i}$

```
 1: Compute the MST T of the point set P.
 2: Set the colour of leaves of T to black, colour red the remaining vertices
 3: Set H ← T.
 4: for each vertex v ∈ T do
 5:     Set c(v) to the degree of v in T.
 6:     Set h(v) ← v.
 7: end for
 8: for each edge e = {u, v} ∉ T, processed in the order of increasing length do
 9:     if h(u) ≠ h(v) then
10:         Let C_e be the cycle closed by {h(u), h(v)} in H.
11:         for each red vertex w ∈ C_e \ {h(u), h(v)} do
12:             c(w) ← c(w) − 1
13:             Output α_{1/c(w)}(w)) = max(r_MST, d(u, v))
14:             if c(w) = 1 then
15:                 Set the colour of w to black.
16:             end if
17:         end for
18:         Remove all edges of C_e from H.
19:         Add a black vertex x_e to H and connect it to all vertices of C_e.
20:         Collapse the all-black component of x_e into a single black vertex x'_e.
21:         Unify h(·) for the vertices of the bi-connected component represented by x'_e.
22:     end if
23: end for
```

- H remains a tree, because each created cycle is replaced by a star graph. The subsequent collapse of the all-black component just reduces the size of the resulting tree.
- Let $e = \{u, v\}$ be the last processed edge. If $h(u) = h(v)$, by induction hypothesis the edge lies within the same block and adding it does not influence any separator vertex. Whenever $h(u) \neq h(v)$, the edge $\{h(u), h(v)\}$ creates a cycle C_e in H; this corresponds to a cycle C'_e in $TUDG(P, d(u, v))$ (since the edges are added in order of increasing length, all shorter edges of $UDG(P, d(u, v))$ have already been processed). The cycle C'_e connects two components for each separator (by induction hypothesis red) vertex lying on this cycle, except the vertices incident to e. This is exactly reflected on line 11 – by induction hypothesis the red vertices processed on line 11 are exactly the separators lying on the cycle C'_e. The invariant is maintained by recoloring red vertices to black on line 15 whenever the separation degree reaches 1, i.e. the vertex stops being a separator. Note also that $h(w)$ remains equal to w while w remains red.
- This invariant is maintained on line 21.

This completes the proof of the lemma. □

From the second point of the previous lemma and from line 13 of $\mathcal{A}\alpha_{1/i}$ we get

Lemma 3. *Algorithm $\mathcal{A}\alpha_{1/i}$ correctly computes values $\alpha_{1/i}(v)$ for $1 \leq i < 5$ and for each vertex $v \in P$.*

Now we will prove that the number of processed edges is of the order of $O(n)$. Let $RNG(P)$ denote the Relative Neighbourhood Graph [9] on the point set P and let $RNG_w(P) = RNG(P \setminus \{w\})$. Recall that the $RNG(P)$ is the geometric graph where P is the set of vertices and two points $u, v \in P$ are connected with a straight line segment if and only if $S(u, d(u, v)) \cap S(v, d(u, v))$ is empty; where $S(x, r)$ denotes an open (i.e. not containing the boundary vertices) sphere/disk centered at vertex x with radius r. The following simple lemma is crucial:

Lemma 4. *Let $e = \{u, v\}$ be the shortest edge connecting two connected components of $TUDG(P, r) \setminus \{w\}$ where $r < d(u, v)$. Then either $e \in RNG(P)$ and it forms angle at least $\pi/3$ with all incident edges of $TUDG(P, r) \setminus \{w\}$, or $e \in RNG_w(P)$ and only the angles $\angle wuv$ or $\angle wvu$ might be smaller than $\pi/3$.*

Proof. If $S(u, d(u, v)) \cap S(v, d(u, v))$ is empty then $e \in RNG(P)$ and the lemma holds. Consider now a vertex $p \in S(u, d(u, v)) \cap S(v, d(u, v))$. Let C_u and C_v denote the components of $TUDG(P, r)$ containing u and v, respectively. $p \notin C_u$, otherwise (p, v) would be the shortest edge connecting C_u and C_v. Analogously $p \notin C_v$. Therefore, the only possibility is $p = w$. After removing w, $S(u, d(u, v)) \cap S(v, d(u, v))$ becomes empty, i.e. $e \in RNG_w(P)$. □

Consider now an edge $e = \{u, v\}$ such that $h(u) \neq h(v)$. By Lemma 2 u and v belong to different blocks of $G' = TUDG(P, d(u', v'))$ where $\{u', v'\}$ is the last edge processed before e. Therefore there exists a separator vertex w such that u and v are in different components of $G' \setminus \{w\}$. Since the edges are processed in the order of increasing length, e must be the shortest edge connecting different components of $G' \setminus \{w\}$. By Lemma 4, $e \in RNG_w(P)$. This means that it is enough to consider on line 8 only the edges from $\bigcup_{w \in P} RNG_w(P)$.

The proofs of the next two lemmas appear in the full paper.

Lemma 5. *The number of edges in $\bigcup_{w \in P} RNG_w(P)$ is in $O(n)$.*

Lemma 6. *Algorithm $\mathcal{A}\alpha_{1/i}$ can be implemented with time complexity $O(n \log n)$.*

Theorem 3 follows directly from Lemma 6 and Lemma 3.

3 Digraph with Max Out-Degree Four

In this section, we prove that given a set of n points in the plane, there is always a strongly connected spanning digraph with max out-degree four and optimal length. In fact, it can be constructed in time $O(n \log n)$.

Theorem 4. *Given n points in the plane, there exists a strongly connected spanning digraph with max degree four and optimal edge length. Furthermore, it can be constructed in $O(n \log n)$ time.*

Proof. Let P be a set of points and T be the Euclidean MST on P. It is known that the max degree of T is six. However, an MST with the same weight as T and max degree five can be obtained with a simple argument. Thus, we assume that T has maximum degree five. Consider the set S of vertices of degree five in T. Let r_4 be the radius obtained from Algorithm $A\alpha_{1/4}$. We compute the set E_v of edges of length at most r_4 as follows: For each vertex $v \in S$ we add to E_v the shortest edge $\{u, v\}$ of length at most r_4 that joins two distinct components of $T \setminus \{v\}$. Clearly $\{u, v\}$ always exists. From Theorems 1 and 2 the optimal range is at least $\alpha_{1/4} = \sigma_{1/4}$. Let $G = T$ and $\overrightarrow{G}$ be the strongly connected graph obtained from orienting in both directions each edge of G. We will add new edges to G in order to form cycles that include every vertex in S. Thus, the max out-degree of $\overrightarrow{G}$ is decreased to four by orienting the cycles in one direction. We will process edges in $\bigcup_{v \in S} E_v$ in descending order according to the hop-length of the cycle that they form with the edges of T.

Let $\{u, w\} \in \bigcup_{v \in S} E_v$ be the edge that forms the longest cycle C. We consider two cases:

- $|C| > 3$. Let $S(x, r)$ denote the open disk centered at x of radius r. Since $\{u, w\}$ is the shortest edge, the lune formed by $S(u, d(u, w)) \cap S(w, d(u, w))$ is empty. Therefore, the angle that $\{u, w\}$ forms with the edges incident to u and w is at least $\pi/3$. Hence, u and w have degree at most four in T. Add $\{u, w\}$ to G and orient C in clockwise order in $\overrightarrow{G}$. Since $\{u, w\}$ is the shortest edge that forms the longest cycle for each vertex in $S \cap C$, we can remove from S all the vertices in C, i.e., $S = S \setminus C$. Observe that vertices in C have out-degree at most four in $\overrightarrow{G}$ and the strong connectivity is not broken.
- $|C| = 3$. Observe that $\{u, v\}, \{w, v\} \in T$. Let $v \in C \setminus \{u, w\}$. Consider the two components G_u, G_w obtained from $G \setminus \{v\}$. Assume that $u \in G_u$ and $w \in G_w$. Since $\{u, w\}$ creates the longest cycle, there does not exist an edge distinct to $\{u, w\}$ in $\bigcup_{x \in S \setminus \{v\}} E_x$ that joins G_u and G_w. However, since $\{u, v\}, \{v, w\}$ are in T, the lune formed by $S(u, d(u, w)) \cap S(w, d(u, w))$ contains v. Therefore, u and w may have degree five in T, since the angle that $\{u, w\}$ forms with $\{u, v\}$ and/or $\{v, w\}$ is less that $\pi/3$.
  Consider the cycle $vu \dots u'v$ in the induced subgraph of $V(G_u) \cup \{v\}$ and the cycle $vw \dots w'v$ in the induced subgraph of $V(G_w) \cup \{v\}$. Let $G = G \cup \{u, w\}$ and remove v from S. Remove $\{v, u\}$ from G if $u \neq u'$. Similarly, remove $\{v, w\}$ from G if $w \neq w'$. Orient the new cycle in $\overrightarrow{G}$ as $vu' \dots uw \dots w'v$. Observe that v has out-degree at most four in $\overrightarrow{G}$ and the out-degree of the vertices in the new cycle does not increase. Furthermore, the strong connectivity is not broken.

When S is empty, $\overrightarrow{G}$ will have max out-degree four. The theorem follows since the strong connectivity of $\overrightarrow{G}$ is never broken. The pseudocode is given in the full paper.

Regarding the time complexity, an Euclidean MST can be computed in $O(n \log n)$ time. From Lemma 5, the number of edges to be processed is $O(n)$.

Thus, it takes $O(n \log n)$ time to sort the edges to be processed. Observe that every vertex is in at most one cycle of length greater than three and/or at most one cycle of length three. Therefore, since each edge is in at most one cycle and there are $O(n)$ edges, the time to complete the construction is $O(n \log n)$. □

4 Digraph with Max Out-Degree Three

In this section we show that given a set of n points, there exists a strongly connected spanning digraph with max out-degree three and length bounded by $2 \cdot \sin(2\pi/9) \cdot \alpha_{1/3}$ which can be constructed in time $O(n \log n)$. We also show that using only the 1/3-weak robust radius, the best approximation of r_3 is at least $2 \cdot \sin(\pi/5) \cdot \alpha_{1/3}$.

Theorem 5. *There exist UDGs that are weak* 1/3*-tough and* $r_3 \geq 2 \cdot \sin(\pi/5) \cdot \alpha_{1/3}$.

Proof. Consider the graph G depicted in Figure 2 with the following properties:

- Each edge has length one.
- The min angle that two edges form with a common neighbour is at least $2\pi/5$
- There is a set $S = \{v_0, v_1, v_2, v'\}$ with degree five such that $\omega(G \setminus S) = 14$.
- There is a path of length at least 2 from v' to v_i and a path of length at least 2 from v_i to $v_{i+1 \mod 3}$.

Observe that the removal of any vertex in G leaves at most three components. Therefore, G is weak 1/3-tough. However, it is not strong 1/3-tough. To create a strongly connected digraph with max degree three it is required to add new edges. However, since the min angle that two edges form with a common neighbour is $2\pi/5$ and each edge has length one, every added edge will be of length at least $2\sin(\pi/5)$. The theorem follows. □

The following Lemma is a simple observation of the construction given in Theorem 4 and is given without proof.

Lemma 7. *Let* $\overrightarrow{G}$ *be the digraph obtained from Theorem 4. For each vertex* v *the angle that* v *forms between out-going edges is at least* $\pi/3$ *and the angle that* v *forms between in-going edges is at least* $\pi/3$.

Theorem 6. *Given* n *points in the plane, there exists a strongly connected spanning digraph with max degree three and edge length bounded by* $2 \cdot \sin(2\pi/9)$ *the optimal edge length. Furthermore, it can be constructed in* $O(n \log n)$ *time.*

Proof. Let P be a set of points and $\overrightarrow{G'}$ be the strongly connected digraph of max-out degree four obtained from Theorem 4. Let $G = G'$ and $\overrightarrow{G} = \overrightarrow{G'}$. Consider the set S of vertices of G such that $\omega(G \setminus \{v\}) = 4$. Let r_3 be the radius of $\alpha_{1/3}$ obtained from Algorithm $A\alpha_{1/3}$. We compute the set E_v of edges of length at most r_3 as follows: For each vertex $v \in S$ we add to E_v the shortest edge $\{u, v\}$

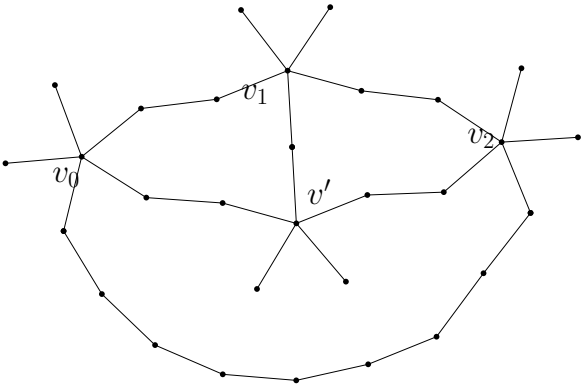


Fig. 2. $\alpha_{1/3} \geq \sigma_{1/3}$

of length at most r_3 that joins two distinct components of $T \setminus \{v\}$. Clearly $\{u, v\}$ always exists. From Lemma 1 the optimal range is at least $\alpha_{1/3}$.

As in Theorem 4 we will add new edges to G in order to form cycles that include every vertex in S. We will prove that the out-degree of the vertices in S can be always decreased without affecting the max out-degree provided that the cycles have length three. However, when the cycles have length greater than three, the orientation of the cycles does not guarantee that every vertex has max out-degree three. Thus, when we orient a cycle of length greater than three, for each vertex v in the cycle that has out-degree four and $\omega(G \setminus \{v\}) = 3$, we will process the shortest edges that connect two components to form cycles of length three, so that the out-degree can be decreased. However, these new edges may have length greater than r_3. We will prove that the length is bounded by $2\sin(2\pi/9) \cdot r_3$.

Firstly we process edges in $\bigcup_{v \in S} E_v$ in descending order according to the hop-length of the cycle that they form with the edges of G'. Let $\{u, w\} \in \bigcup_{v \in S} E_v$ be the edge that forms the longest cycle C. We consider two cases:

- $|C| > 3$. Since $\{u, w\}$ is the shortest edge, the lune formed by $S(u, d(u, w)) \cap S(w, d(u, w))$ is empty. Therefore, the angle that $\{u, w\}$ forms with the edges incident to u and w is at least $\pi/3$. Let $G = G \cup \{u, w\}$ and orient C in clockwise order in $\overrightarrow{G}$. Observe that the orientation of C does not break the connectivity. However, it does not guarantee that the out-degree is decreased to 3. Remove from S each vertex $v \in S \cap C$ such that $\omega(G \setminus \{v\}) \leq 3$.

  Consider the set S' of vertices of out-degree four in $C \setminus S$. For each vertex v' in S', let $E_{v'}$ be the shortest edge $\{u', w'\}$ connecting two distinct components of $G \setminus \{v'\}$ such that $\{v', u'\}$ and $\{v', w'\}$ exist in G'. Add v' to S.

  It remains to prove that $d(u', w') \leq 2\sin(2\pi/9) \cdot r_3$. Let (w_0, v'), $(v', u_0) \in C$ and (v', u_1), (v', u_2) and (v', u_3) be the out-going edges of v' in $\overrightarrow{G}$. From Lemma 7 and the fact that $S(u, d(u, w)) \cap S(w, d(u, w))$ is empty,

$\angle(w_0v'u_0) \geq \pi/3$ and $\angle(u_jv'u_k) \geq \pi/3$. Furthermore, at least two out-going edges $(v', u_a), (v', u_b)$ are in the same component. Therefore, there exist at least two vertices $\{u', w'\}$ in distinct components with angle at most

$$\frac{2\pi - \angle(w_0v'u_0) - \angle(u_av'u_b)}{3} = \frac{4\pi}{9},$$

i.e., $d(u', w') \leq 2\sin(2\pi/9) \cdot r_3$ since the longest edge of G so far is r_3.

- $|C| = 3$. Observe that $\{u, v\}, \{w, v\} \in G'$. Let $v \in C \setminus \{u, w\}$ and G_u, G_w be the two components of $G \setminus v$ such that $u \in G_u$ and $w \in G_w$. Since $\{u, w\}$ creates the longest cycle, there does not exist an edge distinct to $\{u, w\}$ in $\bigcup_{x \in S \setminus \{v\}} E_x$ that joins G_u and G_w. Observe that the out-degree of u and w does not change.

  Consider the cycle $C' = vu \dots u'v$ in the induced subgraph of $V(G_u) \cup \{v\}$ and the cycle $C'' = vw \dots w'v$ in the induced subgraph of $V(G_w) \cup \{v\}$. Let $G = G \cup \{u, w\}$ and remove v from S. Remove $\{v, u\}$ from G if $u \neq u'$. Similarly, remove $\{v, w\}$ from G if $w \neq w'$. Orient the new cycle in $\overrightarrow{G}$ as $vu' \dots uw \dots w'v$. Observe that v has out-degree at most three in $\overrightarrow{G}$ and the out-degree of the vertices in the cycle does not increase. Furthermore, the strong connectivity is not broken.

This process maintains $\overrightarrow{G}$ strongly connected. When S is empty, the max out-degree of $\overrightarrow{G}$ is three.

Regarding the time complexity, from Theorem 4, $\overrightarrow{G}'$ can be computed in $O(n \log n)$ time. From Lemma 5, the number of edges to be processed are of the order of $O(n)$. Thus, it takes $O(n \log n)$ time to sort the edges to be processed. Since every edge is in a constant number of cycles the time to complete the construction is $O(n \log n)$. The theorem follows. □

5 Conclusion

In this paper we studied the problem of how to construct from a set of points in the plane and a positive integer $k > 0$, a geometric strongly connected spanning digraph of out-degree at most k and whose longest edge length is the shortest possible. We proved that the problem can be solved with optimal edge length in polynomial time when the out-degree is at most 4. To quantify the problem we introduced the concept of k-robustness for a UDG radius. We also improved the previous best known upper bound when the out-degree is at most 3. However, it is unknown whether the problem can be solved optimally in polynomial time when the out-degree is at most 3. The problem of finding tighter upper bounds when $k = 2$ also remains open.

References

1. Bauer, D., Broersma, H., Schmeichel, E.: Toughness in graphs–a survey. Graphs and Combinatorics 22(1), 1–35 (2006)

2. Caragiannis, I., Kaklamanis, C., Kranakis, E., Krizanc, D., Wiese, A.: Communication in wireless networks with directional antennae. In: 20th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2008), Munich, Germany, June 14-16, pp. 344–351. IEEE/ACM (2008)
3. Dobrev, S., Kranakis, E., Krizanc, D., Opatrny, J., Ponce, O.M., Stacho, L.: Strong Connectivity in Sensor Networks with Given Number of Directional Antennae of Bounded Angle. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 72–86. Springer, Heidelberg (2010)
4. Francke, A., Hoffmann, M.: The euclidean degree-4 minimum spanning tree problem is np-hard. In: Proceedings of the 25th Annual Symposium on Computational Geometry, pp. 179–188. ACM (2009)
5. Kranakis, E., Krizanc, D., Morales, O.: Maintaining connectivity in sensor networks using directional antennae. In: Nikoletseas, S., Rolim, J. (eds.) Theoretical aspects of Distributed Computing in Sensor Networks, ch. 3, pp. 59–84. Springer (2010) ISBN 978-3-642-14848-4
6. Monma, C., Suri, S.: Transitions in geometric minimum spanning trees. Discrete and Computational Geometry 8(1), 265–293 (1992)
7. Papadimitriou, C.H.: The euclidean travelling salesman problem is np-complete. Theoretical Computer Science 4(3), 237–244 (1977)
8. Parker, R.G., Rardin, R.L.: Guaranteed performance heuristics for the bottleneck traveling salesman problem. Operations Research Letters 2(6), 269–272 (1984)
9. Toussaint, G.T.: The relative neighbourhood graph of a finite planar set. Pattern Recognition 12(4), 261–268 (1980)

Worst-Case Optimal Priority Queues via Extended Regular Counters

Amr Elmasry and Jyrki Katajainen

Department of Computer Science, University of Copenhagen, Denmark

Abstract. We consider the classical problem of representing a collection of priority queues under the operations *find-min*, *insert*, *decrease*, *meld*, *delete*, and *delete-min*. In the comparison-based model, if the first four operations are to be supported in constant time, the last two operations must take at least logarithmic time. Brodal showed that his worst-case efficient priority queues achieve these worst-case bounds. Unfortunately, this data structure is involved and the time bounds hide large constants. We describe a new variant of the worst-case efficient priority queues that relies on extended regular counters and provides the same asymptotic time and space bounds as the original. Due to the conceptual separation of the operations on regular counters and all other operations, our data structure is simpler and easier to describe and understand. Also, the constants in the time and space bounds are smaller.

1 Introduction

A priority queue is a fundamental data structure that maintains a set of elements and supports the operations *find-min*, *insert*, *decrease*, *delete*, *delete-min*, and *meld*. In the comparison-based model, from the $\Omega(n \lg n)$ lower bound for sorting it follows that, if *insert* can be performed in $o(\lg n)$ time, *delete-min* must take $\Omega(\lg n)$ time. Also, if *meld* can be performed in $o(n)$ time, *delete-min* must take $\Omega(\lg n)$ time [1]. In addition, if *find-min* can be performed in constant time, *delete* would not be asymptotically faster than *delete-min*. Based on these observations, a priority queue is said to provide *optimal time bounds* if it can support *find-min*, *insert*, *decrease*, and *meld* in constant time; and *delete* and *delete-min* in $O(\lg n)$ time, where n denotes the number of elements stored.

After the introduction of binary heaps [14], which do not provide optimal time bounds for all priority-queue operations, an important turning point was when Fredman and Tarjan introduced Fibonacci heaps [10]. Fibonacci heaps provide optimal time bounds for all standard operations in the amortized sense. Driscoll et al. [4] introduced run-relaxed heaps, which have optimal time bounds for all operations in the worst case, except for *meld*. On the other side, Brodal [1] introduced meldable priority queues, which provide the optimal worst-case time bounds for all operations, except for *decrease*. Later, by introducing several innovative ideas, Brodal [2] was able to achieve the worst-case optimal time bounds for all operations. Though deep and involved, Brodal's data structure is complicated and should be taken as a proof of existence.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 125–137, 2012.
Springer-Verlag Berlin Heidelberg 2012

Most priority queues supporting worst-case constant *decrease* rely on the concept of *violation reductions*. A *violation* is a node that may, but not necessarily, violate the heap order by being smaller than its parent. When the value of a node is decreased, the node is declared violating. To reduce the number of violations, a violation reduction is performed in constant time whenever possible.

A *numeral system* is a notation for representing numbers in a consistent manner using symbols (*digits*). Operations on these numbers, as increments and decrements of a given digit, must obey the rules governing the numeral system. There is a connection between numeral systems and data-structural design [3,13]. The idea is to relate the number of objects of a specific type in the data structure to the value of a digit. A representation of a number that is subject to increments and decrements of arbitrary digits can be called a *counter*. For an integer $b \geq 2$, a *regular b-ary counter* [3] uses the digits $\{0, \ldots, b\}$ in its representation and imposes the rule that between any two b's there must be a digit other than $b-1$. Such a counter supports increments of arbitrary digits with a constant number of digit changes per operation. An *extended regular b-ary counter* [3,11] uses the digits $\{0, \ldots, b+1\}$ with the constraints that between any two $(b+1)$'s there is a digit other than b, and between any two 0's there is a digit other than 1. Such a counter supports both increments and decrements of arbitrary digits with a constant number of digit changes per operation.

Kaplan and Tarjan [12] (see also [11]) introduced fat heaps as a simplification of Brodal's worst-case optimal priority queues, but these are not meldable in $O(1)$ time. In fat heaps an extended regular ternary counter is used to maintain the trees and an extended regular binary counter to maintain the violation nodes. In a recent study [9], we have simplified fat heaps to work without regular counters.

Our motives for writing the current paper were the following.

1. We simplify Brodal's construction and devise a priority queue that provides optimal worst-case bounds for all operations (2, 3, and 4). Throughout our explanation of the data structure, contrary to [2], we distinguish between the numeral-system operations and other priority-queue operations. The gap between the description complexity of worst-case optimal priority queues and binary heaps [14] is huge. One of our motivations was to narrow this gap.
2. A key ingredient in our construction is the utilization of extended regular binary counters. We describe a strikingly simple implementation of the extended regular counters (5). In spite of their importance for many applications, the existing descriptions [3,11] are sketchy and incomplete.
3. In this paper we are mainly interested in the theoretical performance of the structures discussed. However, some of our ideas may be of practical value.
4. With this paper, we complete our research program on the comparison complexity of priority-queue operations. All the obtained results are summarized in Table 1. Elsewhere, it has been documented that, in practice, worst-case efficient priority queues are often outperformed by simpler non-optimal priority queues. Due to the involved constant factors in the number of element comparisons, this is particularly true if one aims at developing priority queues that achieve optimal time bounds for all the standard operations.

Table 1. The best-known worst-case comparison complexity of different priority-queue operations. A minus sign indicates that the operation is not supported optimally.

Data structure	*find-min*	*insert*	*decrease*	*meld*	*delete-min*
Multipartite priority queues [5]	$O(1)$	$O(1)$	–	–	$\lg n + O(1)$
Two-tier relaxed heaps [6]	$O(1)$	$O(1)$	$O(1)$	–	$\lg n + O(\lg \lg n)$
Meldable priority queues [7]	$O(1)$	$O(1)$	–	$O(1)$	$2 \lg n + O(1)$
Optimal priority queues [this paper, [8]]	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$\approx 70 \lg n$

2 Description of the Data Structure

Let us begin with a high-level description of the data structure. The overall construction is similar to that used in [2]. However, the use of extended regular counters is new. In accordance, the rank rules and hence the tree structures are different from [2]. The set of violation-reduction routines are in turn new.

Each priority queue is composed of two multi-way trees T_1 and T_2, with roots t_1 and t_2 respectively (T_2 can be empty). The atomic components of the priority queues are nodes, each storing a single element. The *rank* of a node x, denoted $rank(x)$, is an integer that is logarithmic in the size (number of nodes) of the subtree rooted at x. The children of a node are stored in a doubly-linked list in non-decreasing rank order. The rank of a subtree is the rank of its root. The trees T_1 and T_2 are heap ordered, except for some violations; but, the element at t_1 is the minimum among the elements of the priority queue. If t_2 exists, the rank of t_1 is less than that of t_2, i.e. $rank(t_1) < rank(t_2)$.

Similar to [2], in the whole data structure there can be up to $O(n)$ violations, not $O(\lg n)$ violations as in run-relaxed heaps and fat heaps. Each violation is guarded by a node that is smaller in value. Any node x, other than t_1, only guards a single list of $O(\lg n)$ violations; these are violations that took place while x was storing the minimum of its priority queue. Such violations must be tackled once the element associated with x is deleted.

For the violations guarded by t_1, a *violation structure* is maintained with each priority queue [2,4,11]. This structure is composed of a resizable array, called the *violation array*, in which the rth entry refers to violations of rank r, and a doubly-linked list, which links the entries of the array that have more than two violations each. The violations guarded by t_1 are divided into two groups: the so-called *active violations* are used to perform violation reductions, and the so-called *inactive violations* are violations whose ranks were larger than the size of the violation array at the time when the violation occurred. All the active violations guarded by t_1, besides being kept in a doubly-linked list, are also kept in the violation structure; all inactive violations are kept in a doubly-linked list. While violations are declared, there are two phases for handling them in accordance to whether the size of the violation array is as big as the largest rank or not. During the first phase the following actions are taken. 1) The new violation is added to the list of inactive violations. 2) The violation array is extended by a constant number of entries. During the second phase the following actions are

taken. 1) The new violation is added to the list of active violations and to the violation structure. 2) A violation reduction is performed if possible.

Handling the violations can be realized by letting each node have one pointer to its violation list, and two pointers to its predecessor and successor in the violation list where the node itself may be in. By maintaining all active violations of the same rank consecutively in the violation list of t_1, the violation array can just have a pointer to the first active violation of any particular rank.

In addition to an element, each node stores its rank and six pointers pointing to: the left sibling, the right sibling (the parent if no right sibling exists), the last child (the rightmost child), the head of the guarded violation list, and the predecessor and the successor in the violation list where the node may be in. To decide whether the right-sibling pointer of a node x points to a sibling or a parent, we locate the node y pointed to by the right-sibling pointer of x and check if the last-child pointer of y points back to x.

Next, we state the rank rules controlling the structure of the multi-way trees:

(a) The rank of a node is one more than the rank of its last child. The rank of a node that has no children is 0.

(b) The *rank sequence* of a node specifies the multiplicities of the ranks of its children. If the rank sequence has a digit d_r, the node has d_r children of rank r. The rank sequences of t_1 and t_2 are maintained in a way that allows adding and removing an arbitrary subtree of a given rank in constant time. This is done by having the rank sequences of those nodes obey the rules of an extended regular binary counter; see 5. When we add a subtree or remove a subtree from *below* t_1 or t_2, we also do the necessary actions to reestablish the constraints imposed by the numeral system.

(c) Consider a node x that is not t_1, t_2, or a child of t_1 or t_2. If the rank of x is r, there must exist at least one sibling of x whose rank is $r-1$, r or $r+1$. Note that this is a relaxation to the rules applied to the rank sequences of t_1 and t_2, for which the same rule also applies [3]. In addition, the number of siblings having the same rank is upper bounded by at most three.

Among the children of a node, there are consecutive subsequences of nodes with consecutive, and possibly equal, ranks. We call each maximal subsequence of such nodes a *group*. By our rank rules, a group has at least two *members*. The difference between the rank of a member of a group and that of another group is at least two, otherwise both constitute the same group.

Lemma 1. *The rank and the number of children of any node in our data structure is $O(\lg n)$, where n is the size of the priority queue.*

Proof. We prove by induction that the size of a subtree of rank r is at least F_r, where F_r is the rth Fibonacci number. The claim is clearly true for $r \in \{0, 1\}$. Consider a node x of rank $r \geq 2$, and assume that the claim holds for all values smaller than r. The last child of x has rank $r-1$. Our rank rules imply that there is another child of rank at least $r-2$. Using the induction hypothesis, the size of these two subtrees is at least F_{r-1} and F_{r-2}. Then, the size of the subtree

rooted at x is at least $F_{r-1} + F_{r-2} = F_r$. Hence, the maximum rank of a node is at most $1.44 \lg n$. By the rank rules, every node has at most three children of the same rank. It follows that the number of children per node is $O(\lg n)$. □

Two trees of rank r can be *joined* by making the tree whose root has the larger value the last subtree of the other. The rank of the resulting tree is $r + 1$. Alternatively, a tree rooted at a node x of rank $r + 1$ can be *split* by detaching its last subtree. If the last group among the children of x now has one member, the subtree rooted at this member is also detached. The rank of x becomes one more than the rank of its current last child. In accordance, two or three trees result from a *split*; among them, one has rank r and another has rank $r-1$, r, or $r+1$. The *join* and *split* operations are used to maintain the constraints imposed by the numeral system. Note that one element comparison is performed with the *join* operation, while the *split* operation involves no element comparisons.

3 Priority-Queue Operations

One complication, also encountered in [2], is that not all violations can be recorded in the violation structure. The reason is that, after a *meld*, the violation array may be too small when the old t_1 with the smaller rank becomes the new t_1 of the melded priority queue. Assume that we have a violation array of size s associated with t_1. The priority queue may contain nodes of rank $r \geq s$. Hence, violations of rank r cannot be immediately recorded in the array and remain inactive. Violation reductions are only performed on active violations whenever possible. Throughout the lifetime of t_1, the array is incrementally extended by the upcoming priority-queue operations until its size reaches the largest rank. Once the array is large enough, no new inactive violations are created. Since each priority-queue operation can only create a constant number of violations, the number of inactive violations is $O(\lg n)$.

In connection with every *decrease* or *meld*, if T_2 exists, a constant number of subtrees rooted at the children of t_2 are removed from below t_2 and added below t_1. Once $\mathit{rank}(t_1) \geq \mathit{rank}(t_2)$, the whole tree T_2 is added below t_1. To be able to move all subtrees from below t_2 and finish the job on time, we should always pick a subtree from below t_2 whose rank equals the current rank of t_1.

The priority queue operations aim at maintaining the following invariants:

1. The minimum is associated with t_1.
2. The second-smallest element is either stored at t_2, at one of the children of t_1, or at one of the violation nodes associated with t_1.
3. The number of entries in the violation list of a node is $O(\lg n)$, assuming that the priority queue that contains this node has n elements.

We are now ready to describe how the priority-queue operations are realized.

find-min(Q)**:** Following the first invariant, the minimum is at t_1.

insert(Q, x)**:** A new node x is given with the value e. If e is smaller than the value of t_1, the roles of x and t_1 are exchanged by swapping the two nodes. The node x is then added below t_1.

meld(Q, Q')**:** This operation involves at most four trees T_1, T_2, T'_1, and T'_2, two for each priority queue; their roots are named correspondingly using lowercase letters. Assume without loss of generality that $value(t_1) \leq value(t'_1)$. The tree T_1 becomes the first tree of the melded priority queue. The violation array of t'_1 is dismissed. If T_1 has the maximum rank, the other trees are added below t_1 resulting in no second tree for the melded priority queue. Otherwise, the tree with the maximum rank among T'_1, T_2, and T'_2 becomes the second tree of the melded priority queue. The remaining trees are added below the root of this tree, and the roots of the added trees are made violating. To keep the number of active violations within the threshold, two violation reductions are performed if possible. Finally, the regular counters that are no longer corresponding to roots are dismissed.

decrease(Q, x, e)**:** The element at node x is replaced by element e. If e is smaller than the element at t_1, the roles of x and t_1 are exchanged by swapping the two nodes (but not their violation lists). If x is either t_1, t_2, or a child of t_1, stop. Otherwise, x is denoted violating and added to the violation structure of t_1; if x was already violating, it is removed from the violation structure where it was in. To keep the number of active violations within the threshold, a violation reduction is performed if possible.

delete-min(Q)**:** By the first invariant, the minimum is at t_1. The node t_2 and all the subtrees rooted at its children are added below t_1. This is accompanied with extending the violation array of T_1, and dismissing the regular counter of T_2. By the second invariant, the new minimum is now stored at one of the children or violation nodes of t_1. By Lemma 1 and the third invariant, the number of minimum candidates is $O(\lg n)$. Let x be the node with the new minimum. If x is among the violation nodes of t_1, a tree that has the same rank as x is removed from below t_1, its root is made violating, and is attached in place of the subtree rooted at x. If x is among the children of t_1, the tree rooted at x is removed from below t_1. The inactive violations of t_1 are recorded in the violation array. The violations of x are also recorded in the array. The violation list of t_1 is appended to that of x. The node t_1 is then deleted and replaced by the node x. The old subtrees of x are added, one by one, below the new root. To keep the number of violations within the threshold, violation reductions are performed as many times as possible. By the third invariant, at most $O(\lg n)$ violation reductions are to be performed.

delete(Q, x)**:** The node x is swapped with t_1, which is then made violating. To remove the current root x, the same actions are performed as in *delete-min*.

In our description, we assume that it is possible to dismiss an array in constant time. We also assume that the doubly-linked list indicating the ranks where a reduction is possible is realized inside the violation array, and that a regular counter is compactly represented within an array. Hence, the only garbage created by freeing a violation structure or a regular counter is an array of pointers. If it is not possible to dismiss such an array in constant time, we rely on incremental garbage collection. In that case, to dismiss a violation structure or a regular counter, we add it to the garbage pile, and release a constant amount of garbage in connection with every priority-queue operation. It is not hard to prove by induction that the

sum of the sizes of the violation structures, the regular counters, and the garbage pile remains linear in the size of the priority queue.

4 Violation Reductions

Each time when new violations are introduced, we perform equally many violation reductions whenever possible. A violation reduction is possible if there exists a rank recording at least three active violations. This will fix the maximum number of active violations at $O(\lg n)$. Our violation reductions diminish the number of violations by either getting rid of one violation, getting rid of two and introducing one new violation, or getting rid of three and introducing two new violations. We use the powerful tool that the rank sequence of t_1 obey the rules of a numeral system, which allows adding a new subtree and removing a subtree of a given rank from below t_1 in worst-case constant time. When a subtree with a violating root is added below t_1, its root is no longer violating.

One consequence of allowing $O(n)$ violations is that we cannot use the violation reductions exactly in the form described for run-relaxed heaps or fat heaps. When applying the cleaning transformation to a node of rank r (see [4] for the details), we cannot any more be sure that its sibling of rank $r+1$ is not violating, simply because there can be violations that are guarded by other nodes. We then have to avoid executing the cleaning transformation by the violation reductions.

Let x_1, x_2, and x_3 be three violations of the same rank r. We distinguish several cases to be considered when applying our reductions:

Case 1. If, for some $i \in \{1, 2, 3\}$, x_i is neither the last nor the second-last child, detach the subtree rooted at x_i from its parent and add it below t_1. The node x_i will not be anymore violating. The detachment of x_i may leave one or two groups with one member (but not the last group). If this happens, the subtree rooted at each of these singleton members is then detached and added below t_1. (We can still detach the subtree of x_i even when x_i is one of the last two children of its parent, conditioned that such detachment leaves this last group with at least two members and retains the rank of the parent.)

For the remaining cases, after checking Case 1, we assume that each of x_1, x_2, and x_3 is either the last or the second-last child of its parent. Let s_1, s_2, and s_3 be the other member of the last two members of the groups of x_1, x_2, and x_3, respectively. Let p_1, p_2, and p_3 be the parents of x_1, x_2, and x_3, respectively. Assume without loss of generality that $rank(s_1) \geq rank(s_2) \geq rank(s_3)$.

Case 2. $rank(s_1) = rank(s_2) = r+1$, or $rank(s_1) = rank(s_2) = r$, or $rank(s_1) = r$ and $rank(s_2) = r - 1$:

(a) $value(p_1) \leq value(p_2)$: Detach the subtrees rooted at x_1 and x_2, and add them below t_1; this reduces the number of violations by two. Detach the subtree rooted at s_2 and attach it below p_1 (retain rank order); this does not introduce any new violations. Detach the subtree rooted at the last child of p_2 if it is a singleton member, detach the remainder of the

subtree rooted at p_2, change the rank of p_2 to one more than that of its current last child, and add the resulting subtrees below t_1. Remove a subtree with the old rank of p_2 from below t_1, make its root violating, and attach it in the old place of the subtree rooted at p_2.

(b) $value(p_2) < value(p_1)$: Change the roles of x_1, s_1, p_1 and x_2, s_2, p_2, and apply the same actions as in Case 2(a).

Case 3. $rank(s_1) = r + 1$ and $rank(s_2) = r$:

(a) $value(p_1) \leq value(p_2)$: Apply the same actions as in Case 2(a).

(b) $value(p_2) < value(p_1)$: Detach the subtrees rooted at x_1 and x_2, and add them below t_1; this reduces the number of violations by two. Detach the subtree rooted at s_2 if it becomes a singleton member of its group, detach the remainder of the subtree rooted at p_2, change the rank of p_2 to one more than that of its current last child, and add the resulting subtrees below t_1. Detach the subtree rooted at s_1, and attach it in the old place of the subtree rooted at p_2; this does not introduce any new violations. Detach the subtree rooted at the current last child of p_1 if such child becomes a singleton member of its group, detach the remainder of the subtree rooted at p_1, change the rank of p_1 to one more than that of its current last child, and add the resulting subtrees below t_1. Remove a subtree of rank $r + 2$ from below t_1, make its root violating, and attach it in the old place of the subtree rooted at p_1.

Case 4. $rank(s_1) = rank(s_2) = rank(s_3) = r - 1$:

Assume without loss of generality that $value(p_1) \leq \min\{value(p_2), value(p_3)\}$. Detach the subtrees of x_1, x_2, and x_3, and add them below t_1; this reduces the number of violations by three. Detach the subtrees of s_2 and s_3, and join them to form a subtree of rank r. Attach the resulting subtree in place of x_1; this does not introduce any new violations. Detach the subtree rooted at the current last child of each of p_2 and p_3 if such child becomes a singleton member of its group, detach the remainder of the subtrees rooted at p_2 and p_3, change the rank of each of p_2 and p_3 to one more than that of its current last child, and add the resulting subtrees below t_1. Remove two subtrees of rank $r + 1$ from below t_1, make the roots of each of them violating, and attach them in the old places of the subtrees rooted at p_2 and p_3.

Case 5. $rank(s_1) = r + 1$ and $rank(s_2) = rank(s_3) = r - 1$:

(a) $value(p_1) \leq \min\{value(p_2), value(p_3)\}$: Apply same actions as Case 4.

(b) $value(p_2) < value(p_1)$: Apply the same actions as in Case 3(b).

(c) $value(p_3) < value(p_1)$: Change the roles of x_2, s_2, p_2 to x_3, s_3, p_3, and apply the same actions as in Case 3(b).

The following properties are the keys for the success of our violation-reduction routines. 1) Since there is no tree T_2 when a violation reduction takes place, the rank of t_1 will be the maximum rank among all other nodes. In accordance, we can remove a subtree of any specified rank from below t_1. 2) Since t_1 has the minimum value of the priority queue, its children are not violating. In accordance, we can add a subtree below t_1 and ensure that its root is not violating.

5 Extended Regular Binary Counters

An extended regular binary counter represents a non-negative integer n as a string $\langle d_0, d_1, \ldots, d_{\ell-1} \rangle$ of digits, least-significant digit first, such that $d_i \in \{0, 1, 2, 3\}$, $d_{\ell-1} \neq 0$, and $n = \sum_i d_i \cdot 2^i$. The main constraint is that every 3 is preceded by a 0 or 1 possibly having any number of 2's in between, and that every 0 is preceded by a 2 or 3 possibly having any number of 1's in between. This constraint is stricter than the standard one, which allows the first 3 and the first 0 to come after any (or even no) digits. An extended regular counter [3,11] supports the increments and decrements of arbitrary digits with a constant number of digit changes per operation.

Brodal [2] showed how, what he called, a *guide* can realize a regular binary counter (the digit set has three symbols and the counter supports *increments* in constant time). To support *decrements* in constant time as well, he suggested to couple two such guides back to back, and accordingly used six symbols altogether. We show how to implement, in a simpler way, an extended regular binary counter more efficiently. The construction described here was sketched in [11]; our description is more detailed and more precise.

We partition any sequence into *blocks* of consecutive digits, and digits that are in no blocks. We have two categories of blocks: blocks that end with a 3 are of the forms $12^*3, 02^*3$, and blocks that end with a 0 are of the forms $21^*0, 31^*0$ (we assume that least-significant digits come first, and * means zero or more repetitions). We call the last digit of a block the *distinguishing digit*, and the other digits of the block the *block members*. Note that the distinguishing digit of a block may be the first member of a block from the other category.

To efficiently implement *increment* and *decrement* operations, a *fix* is performed at most twice per operation. A fix does not change the value of a number. When a digit that is a member of a block is increased or decreased by one, we may need to perform a fix on the distinguishing digit of its block. We associate a forward pointer f_i with every digit d_i, and maintain the invariant that all the members of the same block point to the distinguishing digit of that block. The forward pointers of the digits that are not members of a block point to an arbitrary digit. Starting from any block member, we can access the distinguishing digit of its block, and hence perform the required fix, in constant time.

As a result of an *increment* or a *decrement*, the following properties make such a construction possible.

- A block may only extend from the beginning and by only one digit. In accordance, the forward pointer of this new block member inherits the same value as the forward pointer of the following digit.
- A newly created block will have only two digits. In accordance, the forward pointer of the first digit is made to point to the other digit.
- A fix that is performed unnecessarily is not harmful (keeps the representation regular). In accordance, if a block is destroyed when fixing its distinguishing digit, no changes are done with the forward pointers.

A string of length zero represents the number 0. In our pseudo-code, the change in the length of the representation is implicit; the key observation is that the length can only increase by at most one digit with an *increment* and decrease by at most two digits with a *decrement*.

For $d_j = 3$, a *fix-carry* for d_j is performed as follows:

Algorithm *fix-carry*$(\langle d_0, d_1, \ldots, d_{\ell-1}\rangle, j)$

assert $0 \leq j \leq \ell - 1$ **and** $d_j = 3$
$d_j \leftarrow 1$
increase d_{j+1} by 1
if $d_{j+1} = 3$
    $f_j \leftarrow j + 1$ // a new block of two digits
else
    $f_j \leftarrow f_{j+1}$ // extending a *possible* block from the beginning

As a result of a *fix-carry* the value of a number does not change. Accompanying a *fix-carry* two digits are changed. In the corresponding data structure, this results in performing a *join*, which involves one element comparison.

The following pseudo-code summarizes the actions needed to increase the ith digit of a number by one.

Algorithm *increment*$(\langle d_0, d_1, \ldots, d_{\ell-1}\rangle, i)$

assert $0 \leq i \leq \ell$
if $d_i = 3$
    fix-carry$(\langle d_0, d_1, \ldots, d_{\ell-1}\rangle, i)$ // either this fix is executed
$j \leftarrow f_i$
if $j \leq \ell - 1$ **and** $d_j = 3$
    fix-carry$(\langle d_0, d_1, \ldots, d_{\ell-1}\rangle, j)$
increase d_i by 1
if $d_i = 3$
    fix-carry$(\langle d_0, d_1, \ldots, d_{\ell-1}\rangle, i)$ // or this fix

Using case analysis, it is not hard to verify that this operation maintains the regularity of the representation.

For $d_j = 0$, a *fix-borrow* for d_j is performed as follows:

Algorithm *fix-borrow*$(\langle d_0, d_1, \ldots, d_{\ell-1}\rangle, j)$

assert $0 \leq j < \ell - 1$ **and** $d_j = 0$
$d_j \leftarrow 2$
decrease d_{j+1} by 1
if $d_{j+1} = 0$
    $f_j \leftarrow j + 1$ // a new block of two digits
else
    $f_j \leftarrow f_{j+1}$ // extending a *possible* block from the beginning

As a result of a *fix-borrow* the value of a number does not change. Accompanying a *fix-borrow* two digits are changed. In the corresponding data structure, this results in performing a *split*, which involves no element comparisons.

The following pseudo-code summarizes the actions needed to decrease the ith digit of a number by one.

Algorithm *decrement*$(\langle d_0, d_1, \ldots, d_{\ell-1}\rangle, i)$

```
1:  assert 0 ≤ i ≤ ℓ − 1
2:  if d_i = 0
3:     fix-borrow(⟨d_0, d_1, ..., d_{ℓ−1}⟩, i)   // either this fix is executed
6:  j ← f_i
7:  if j < ℓ − 1 and d_j = 0
8:     fix-borrow(⟨d_0, d_1, ..., d_{ℓ−1}⟩, j)
8:  decrease d_i by 1
10: if i < ℓ − 1 and d_i = 0
11:    fix-borrow(⟨d_0, d_1, ..., d_{ℓ−1}⟩, i)   // or this fix
```

Using case analysis, it is not hard to verify that this operation maintains the regularity of the representation.

In our application, the outcome of splitting a tree of rank $r+1$ is not always two trees of rank r as assumed above. The outcome can also be one tree of rank r and another of rank $r+1$; two trees of rank r and a third tree of a smaller rank; or one tree of rank r, one of rank $r-1$, and a third tree of a smaller rank. We can handle the third tree, if any, by adding it below t_1 (executing an *increment* just after the *decrement*). The remaining two cases, where the *split* results in one tree of rank r and another of rank either $r+1$ or $r-1$, are pretty similar to the case where we have two trees of rank r. In the first case, we have detached a tree of rank $r+1$ and added a tree of rank r and another of rank $r+1$; this case maintains the representation regular. In the second case, we have detached a tree of rank $r+1$ and added a tree of rank r and another of rank $r-1$; after that, there may be three or four trees of rank $r-1$, and one *join* (executing a *fix-carry*) at rank $r-1$ may be necessary to make the representation regular. In the worst case, a *fix-borrow* may require three element comparisons: two because of the extra addition (*increment*) and one because of the extra *join*. A *decrement*, which involves two fixes, may then require up to six element comparisons.

6 Conclusions

We showed that a simpler data structure achieving the same asymptotic bounds as Brodal's data structure [2] exists. We were careful not to introduce any artificial complications when presenting our data structure. Our construction reduces the constant factor hidden behind the big-Oh notation for the worst-case number of element comparisons performed by *delete-min*. Theoretically, it would be interesting to know how low such factor can get.

Our storage requirements are as follows. Every node stores an element, a rank, two sibling pointers, a last-child pointer, a pointer to its violation list, and two pointers for the violation list it may be in. The violation structure and the regular counters require logarithmic space. The good news is that we save four pointers per node in comparison with [2]. The bad news is that, for example, binomial queues [13] can be implemented with only two pointers per node.

We summarize the main differences between our construction and that in [2].

+ We use a standard numeral system with fewer digits (four instead of six). Besides improving the constant factors, this allows for the logical distinction between the operations of the numeral system and other operations.
+ We use normal joins instead of three-way joins, each involving one element comparison instead of two.
+ We do not use parent pointers, except for the last children. This saves one pointer per node and allows swapping of nodes in constant time. Since node swapping is possible, elements can be stored directly inside nodes; this saves two more pointers per node.
+ We gathered the violations associated with every node in one violation list, instead of two; this saves one more pointer per node.
+ We bound the number of violations by restricting their total number to be within a threshold, whereas the treatment in [2] imposes an involved numeral system that constrains the number of violations per rank.
− We check more cases within our violation reductions.
− We have to deal with larger ranks; the maximum rank may go up to $1.44 \lg n$, instead of $\lg n$.

References

1. Brodal, G.S.: Fast Meldable Priority Queues. In: Sack, J.-R., Akl, S.G., Dehne, F., Santoro, N. (eds.) WADS 1995. LNCS, vol. 955, pp. 282–290. Springer, Heidelberg (1995)
2. Brodal, G.S.: Worst-Case Efficient Priority Queues. In: 7th ACM-SIAM Symposium on Discrete Algorithms, pp. 52–58. ACM/SIAM, New York/Philadelphia (1996)
3. Clancy, M.J., Knuth, D.E.: A Programming and Problem-Solving Seminar. Report STAN-CS-77-606. Computer Science Department, Stanford University, Stanford (1977)
4. Driscoll, J.R., Gabow, H.N., Shrairman, R., Tarjan, R.E.: Relaxed Heaps: An Alternative to Fibonacci Heaps with Applications to Parallel Computation. Commun. ACM 31(11), 1343–1354 (1988)
5. Elmasry, A., Jensen, C., Katajainen, J.: Multipartite Priority Queues. ACM Trans. Algorithms 5(1), Article 14 (2008)
6. Elmasry, A., Jensen, C., Katajainen, J.: Two-tier Relaxed Heaps. Acta Inform. 45(3), 193–210 (2008)
7. Elmasry, A., Jensen, C., Katajainen, J.: Strictly-Regular Number System and Data Structures. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 26–37. Springer, Heidelberg (2010)

8. Elmasry, A., Katajainen, J.: Worst-Case Optimal Priority Queues Via Extended Regular Counters, Ithaca, E-print arXiv:1112.0993. arXiv.org (2011)
9. Elmasry, A., Katajainen, J.: Fat Heaps without Regular Counters. In: Rahman, M. S., Nakano, S.-I. (eds.) WALCOM 2012. LNCS, vol. 7157, pp. 173–185. Springer, Heidelberg (2012)
10. Fredman, M.L., Tarjan, R.E.: Fibonacci Heaps and their Uses in Improved Network Optimization Algorithms. J. ACM 34(3), 596–615 (1987)
11. Kaplan, H., Shafrir, N., Tarjan, R.E.: Meldable Heaps and BOolean Union-Find. In: 34th Annual ACM Symposium of Theory of Computing, pp. 573–582. ACM, New York (2002)
12. Kaplan, H., Tarjan, R.E.: New Heap Data Structures. Report TR-597-99. Department of Computer Science, Princeton University, Princeton (1999)
13. Vuillemin, J.: A Data Structure for Manipulating Priority Queues. Commun. ACM 21(4), 309–315 (1978)
14. Williams, J.W.J.: Algorithm 232: Heapsort. Commun. ACM 7(6), 347–348 (1964)

The Complexity of Minor-Ancestral Graph Properties with Forbidden Pairs

Eli Fox-Epstein[1] and Danny Krizanc[2]

[1] Tufts University
[2] Wesleyan University

Abstract. Robertson and Seymour (in work starting with [15]) demonstrated that any minor-ancestral graph property can be decided in polynomial time. Lewis and Yannakakis [14] showed that for any nontrivial node-hereditary graph property, the problem of given a graph, finding the size of the largest induced subgraph of the graph that has the property, is **NP**-hard. In this paper, we completely characterize those minor-ancestral properties for which the problem of deciding if a given graph contains a subgraph with the property that respects a given set of forbidden vertex pairs (i.e., if one vertex from a pair is in the subgraph then the other isn't) is in **P** and for which such properties the problem is **NP**-complete. In particular, we show that if a given minor-ancestral property can be characterized by the containment of one of a finite set of graphs as a subgraph, the corresponding decision problem with forbidden vertex pairs is in **P**, otherwise its **NP**-complete. Unfortunately, we further show that the problem of deciding if a minor-ancestral property (presented as a set of characteristic minors) can be so characterized is **NP**-hard. Finally we observe that a similar characterization holds for the case of finding subgraphs satisfying a set of forbidden edge pairs and that our problems are all fixed parameter tractable.

1 Introduction

The study of the computational complexity of deciding graph properties closed under various natural operations is well-established. For example, it is well-known that graph properties closed under taking minors can be decided in polynomial time [16]. In the same vein, optimizing the number of nodes one must delete from a graph to obtain one with a given property is **NP**-hard for any graph property closed under taking induced subgraphs [14]. Occasionally, one finds that polynomial time graph properties become hard when natural constraints are placed on the witnesses to the property. E.g., finding a path between nodes in a graph is easy when unconstrained but becomes **NP**-hard when the path must satisfy some constraints upon which pairs of vertices may be chosen [9]. In this paper, we characterize the computational complexity of deciding minor-ancestral graph properties under such forbidden vertex pairs constraints.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 138–147, 2012.

A graph property Π is a subset of the set of all finite, unlabeled,[1] undirected graphs.[2] A property is *trivial* if it or its complement is finite. If $G \in \Pi$, we say G *satisfies* Π, $\Pi(G)$ *holds* or just $\Pi(G)$.

Given a graph $G = (V_G, E_G)$ and an edge $uv \in E_G$, a *subdivision* is an operation that replaces the edge uv with an intermediate vertex $w \notin V_G$ and edges uw and wv. A graph H is a subdivision of G if it can be obtained by zero or more subdivisions of G. We define subdividing an edge k times to be the operation of replacing the edge with a path of length $k+1$ between the endpoints of the edge. An *edge contraction* of an edge $uv \in E_G$ replaces uv and vertices u and v with a new vertex w and edges $\{wx \mid ux \in E_G \vee vx \in E_G\}$. H is a *minor* of G if one can perform zero or more contractions on a subgraph of G to obtain H. H is a *topological minor* of G if a subgraph of G is a subdivision of H.

A graph property Π is *minor-hereditary* if given graphs G and H with H a minor of G, $\Pi(G)$ implies $\Pi(H)$, i.e, the property is closed under taking minors. Examples of such properties include planar, acyclic, bounded treewidth, etc. The complement of a minor-hereditary property is *minor-ancestral*. A graph property Π is *node-hereditary* if given graphs G and H with H an induced subgraph of G, $\Pi(G)$ implies $\Pi(H)$, i.e., the property is closed under taking induced subgraphs. Examples include planar, outerplanar, bipartite, acyclic, interval graph, chordal, etc. Finally, a graph property Π is *subgraph-hereditary* if given graphs G and H with H a subgraph of G, $\Pi(G)$ implies $\Pi(H)$, i.e., the property is closed under taking subgraphs. Note that any minor-hereditary (node-hereditary or subgraph-hereditary) property Π is characterized by a set (possibly infinite), C_Π, of *forbidden* minors (respectively, induced subgraphs or subgraphs), i.e., any graph containing one of the graphs in C_Π as a minor (respectively, induced subgraph or subgraph) does not have the property Π. Similarly, a minor-ancestral property, Π, is characterized by a set of graphs C_Π such that any graph with the property must contain some graph in C_Π as a minor. FSP (for *Forbidden Subgraph Property*) is the class of subgraph-hereditary properties characterized by a finite set of forbidden subgraphs [5]. We define co-FSP as the set of graph properties that are the complement of some property in FSP. Note that properties Π in co-FSP are characterized by a finite set of graphs C_Π such that any graph with property Π must contain some graph in C_Π as a subgraph.

In a series of papers starting with [15], Robertson and Seymour established the graph minor theorem, that the partial ordering of graphs under minors forms a well-quasi-ordering. Among the implications of their work is that minor-hereditary graph properties are characterized by a finite set of forbidden minors which in turn they showed implies that any minor-hereditary (or minor-ancestral) property can be decided in polynomial time [16]. This discovery has lead to a large number of results showing various graph problems are in **P**. (See [8] for a start.)

[1] That is, isomorphic graphs are identified with each other.

[2] For the most part, we follow standard graph-theoretic notation, as found in any standard text on the subject such as [3].

Building on the work of Krishnamoorthy and Deo [13], Lewis and Yannakakis [14] showed that for any nontrivial node-hereditary property the *node deletion* problem, i.e., the problem of given a graph G, finding the size of a largest induced subgraph of G with the property, is **NP**-hard. Research along these lines for different graph properties and graph operations continues. For example, Asano and Hirata [2] show that for minor-hereditary properties with characteristic forbidden minors that are all made up of 3-connected components, the edge deletion problem (finding the size of the largest subgraph with the property) is **NP**-complete. Recently, Alon et al. [1] showed (among other results) that if all bipartite graphs satisfy an subgraph-hereditary property, then the edge deletion problem for that property is **NP**-hard even to approximate.

The concept of forbidden pairs of vertices comes from the field of software testing and automated analysis, and was introduced in [12]. Additional motivation for forbidden vertex pairs problems comes from computational biology by Brady et al. [4]. Gabow, Maheshwari, and Osterweil [9] proved that the problem of deciding if a path that respects forbidden vertex pairs exists between two vertices is **NP**-complete. Since then, a series of papers [18,11] has shown that depending upon different restrictions one may place on the set of forbidden vertex pairs, the problem may become polynomial-time solvable.

Childs and Kothari [5] introduced the class FSP of subgraph-hereditary properties characterized by a finite set of forbidden subgraphs in their study of the quantum query complexity of minor-hereditary properties. They show that minor-hereditary properties that are in FSP can be solved using $o(n^{3/2})$ quantum queries (for graphs with n vertices) while those not in FSP require $\Theta(n^{3/2})$ queries. co-FSP is particularly relevant to our study of minor-ancestral properties.

The results of Robertson and Seymour show that minor-ancestral properties are "easy" (i.e., in polynomial time) while the results of Lewis and Yannakakis imply that certain optimization problems for node-hereditary properties are "hard" (i.e., **NP**-hard). Gabow et al. show that a particular easy problem (*st*-connectivity) becomes hard when a set of forbidden vertex pairs is added to the problem. In this paper, we investigate when minor-ancestral properties move from being easy (a la Robertson and Seymour) to being hard (a la Lewis and Yannakakis) when forbidden vertex pairs are added. In particular, we give a complete characterization of when deciding a minor-ancestral property with forbidden vertex pairs is in **P** and when it is **NP**-hard. As it turns out, our characterization is based upon the same dichotomy observed by Childs and Kothari in their study of the quantum query complexity of minor-hereditary properties.

Given a graph property Π we define the forbidden vertex pairs decision problem for Π, $\text{FVP}_\Pi(G, F)$, as follows: The input is a graph $G = (V_G, E_G)$ and a set of vertex pairs, $F \subset V_G \times V_G$, and the output is 'yes' if and only if there exists H a subgraph of G such that $\Pi(H)$, and $V_H \times V_H$ and F are disjoint. For a given H and F we say H *respects* F if $V_H \times V_H$ and F are disjoint.

Our main result completely characterizes the complexity of the forbidden vertex pairs decision problem for minor-ancestral graph properties. In particular, suppose Π is a nontrivial minor-ancestral graph property. If Π is in co-FSP then

FVP_Π is in **P**. If not, it is in **NP**-complete. In other words, if a graph property can be characterized by the containment of one of a finite number of subgraphs, then its corresponding forbidden vertex pairs problem is in **P**; otherwise, it is **NP**-complete.

The rest of the paper is organized as follows. In the next section we prove our main result. This is followed by a discussion of consequences and extensions of the main theorem along with our proof that deciding if a minor-ancestral property (presented as a set of characteristic minors) is in co-FSP is **NP**-hard. We conclude with some natural extensions and discussion of open problems.

2 Main Result

In this section we prove our main result. It follows from a series of lemmas shown below. We define a *dangling path* as one in which zero or more degree-two vertices are terminated by a degree-one vertex.

Lemma 1. *If Π is minor-ancestral but not in co-FSP then there exists a graph $G \in \Pi$ containing edge uw such that replacing uw with dangling paths attached to each of u and w produces a graph $G' \notin \Pi$.*

Proof. $\overline{\Pi}$ is minor-hereditary and not in FSP. By Lemma 3.5 in [5], there exists a graph G not in $\overline{\Pi}$ and an edge $uw \in E_G$ such that one can replace the edge with dangling paths attached to each of u and w and produce a graph $G' \in \overline{\Pi}$. Since $G \notin \overline{\Pi}$, we have $G \in \Pi$. Furthermore, replacing uw with dangling paths from u and w produces a graph $G' \notin \Pi$. □

Lemma 2. *If Π is a graph property in co-FSP then* FVP_Π *is in **P**.*

Proof. The subgraph isomorphism problem is known to be in **P** when one input is fixed. As Π is in co-FSP, there exists a fixed set of graphs, C_Π such that any member of Π will contain at least one of the graphs in C_Π as a subgraph. We can find all copies of the graphs in C_Π that appear as subgraphs of a given input graph to FVP_Π in time polynomial in the size of the input graph (the polynomial depending upon the size of the largest graph in C_Π which is fixed independent of the size of the input). Each such copy can be checked to see if it respects the forbidden vertex pairs in time polynomial in the size of the forbidden vertex pairs set. If a subgraph is found respecting the forbidden vertex pairs then the input satisfies Π, otherwise it does not. Thus, FVP_Π can be decided in polynomial time. □

Lemma 3. *Let Π be a minor-ancestral property not in co-FSP. Then* FVP_Π *is **NP**-complete.*

Proof. To demonstrate the **NP**-completeness of FVP_Π, we will first show that it is in **NP**. Π is minor-ancestral so there exists a finite C_Π. Given a certificate consisting of a graph G, a minor H in C_Π, and a list of contractions, one can verify that G is a subgraph of the input graph that respects the forbidden vertex pairs, and that performing the contractions on G lead to H.

Next, to show **NP**-hardness, we reduce Monotone One-in-Three Sat (well-known to be **NP**-complete, see [17]) to the forbidden vertex pairs problem for Π. The problem takes as input a formula in 3-CNF form where all literals are variables (no negations) and accepts if there is a satisfying assignment with exactly one true literal per clause. Let $X = (l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge ... \wedge (l_{n,1} \vee l_{n,2} \vee l_{n,3})$ be input for an instance of Monotone One-in-Three Sat.

By Lemma 1, there exists a graph $G \in \Pi$ with an edge uw such that deleting the edge and adding dangling paths to u and to w produces a graph not in Π. Note that G and uw depend only on property and remain constant with respect to X. From G, we construct a graph G' and a set of forbidden vertex pairs to be the input to FVP_Π. An affirmative decision will occur exactly when X is satisfiable. Graph G' is generated from G as follows:

1. Delete edge uw.
2. Add disjoint vertices $v_{1,1}, v_{1,2}, v_{1,3}, ..., v_{n,1}, v_{n,2}, v_{n,3}$. Call these v-vertices.
3. For each $1 \leq i < n$ and $1 \leq j, k \leq 3$, add edges $v_{i,j}v_{i+1,k}$.
4. Add edges $uv_{1,j}$ and $wv_{n,j}$ for $1 \leq j \leq 3$.

An example of our construction is given in Figure 1.

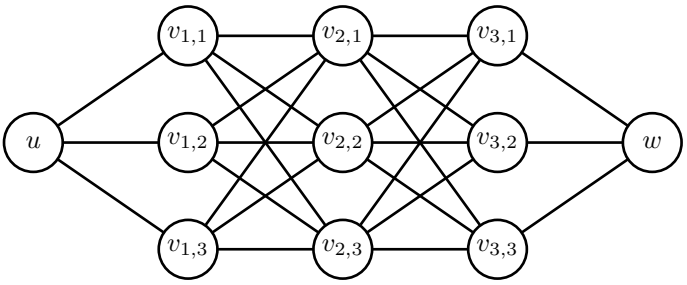


Fig. 1. Example construction between u and w for a 3-clause instance of Monotone 1-in-3 Sat

The forbidden vertex pairs F are comprised of two types:

Intraclause: For all $1 \leq j < k \leq 3$ and $1 \leq i \leq n$, forbid $v_{i,j}$ with $v_{i,k}$.
Interclause: If $l_{i,j} = l_{x,y}$, forbid $v_{i,j}$ with $v_{x,z}$ for $1 \leq z \neq y \leq 3$.

As an example, if $X = (l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3})$ with $l_{1,1} = l_{2,2}$, the forbidden pairs would be

$$\{(v_{1,1}, v_{1,2}), (v_{1,1}, v_{1,3}), (v_{1,2}, v_{1,3}), (v_{2,1}, v_{2,2}),$$
$$(v_{2,1}, v_{2,3}), (v_{2,2}, v_{2,3}), (v_{1,1}, v_{2,1}), (v_{1,1}, v_{2,3}),$$
$$(v_{1,2}, v_{2,2}), (v_{1,3}, v_{2,2})\}.$$

We now show that there is a subgraph of G' that respects F and is a subdivision of G if and only if X is satisfiable. Consider a subgraph of G' that includes

all of the original vertices and edges of G except uw and for each i, includes one vertex $v_{i,j}$ with $1 \leq j \leq 3$. One can form a mapping between any such subgraph and a truth assignment to the variables in X: for all $v_{i,j} \in V_{G'}$, $l_{i,j}$ is true and all other literals are false. We will show that the only such subgraphs respecting the forbidden vertex pairs are those that produce satisfying assignments of X.

Suppose that X is satisfiable. We will show that $\text{FVP}_\Pi(G', F)$ is true. Consider the subgraph induced by deleting any v-vertex $v_{i,j}$ where $l_{i,j}$ is false. Since for each i, there exists a j such that $l_{i,j}$ is true, there is a path from u to w through the v-vertices, so the resulting graph is a subdivision of G. Furthermore, since there is exactly one per clause, no two vertices are forbidden due to intraclause forbidden vertex pairs. Whenever $l_{i,j} = l_{x,y}$, $v_{i,j}$ and $v_{x,y}$ will both either be in or be excluded from the subgraph, so no two vertices will be forbidden due to the interclause forbidden vertex pairs. Therefore, when X is satisfiable, $\text{FVP}_\Pi(G', F)$ is true.

Now, suppose that $\text{FVP}_\Pi(G', F)$ is true. There exists a subgraph G'' of G' that respects the forbidden vertex pairs and has G as a minor. There must be a path from u to w through the v-vertices remaining in G'' since by Lemma 1, if one replaces uw with dangling paths on u and w, the resulting graph is not in Π. By construction, for each $1 \leq i \leq n$, one (and exactly one by the forbidden vertex pairs) $v_{i,j}$ must be a member of this path. The corresponding truth assignment must satisfy X, as exactly one literal per clause is true and the interclause forbidden vertex pairs ensure consistency (i.e., x_i is not declared to be true in one clause but false in another). □

Theorem 1. *If minor-ancestral property Π is in co-FSP then* FVP_Π *is in* ***P****, otherwise* FVP_Π *is* ***NP****-complete.*

Proof. If Π is in co-FSP then Lemma 2 shows that FVP_Π is in **P**. Otherwise, Lemma 3 shows that FVP_Π is **NP**-complete. □

3 Discussion

As an immediate consequence of our main result we are able to determine the complexity of deciding a large number of graph properties under forbidden vertex pairs constraints. For example, deciding if a graph has a cycle or nonplanar subgraph respecting a set of forbidden vertex pairs is **NP**-complete while deciding if a graph has a triangle or a path of fixed length k respecting a set of forbidden vertex pairs is in **P**. We further observe that the proof of Lemma 3 is easily adapted to prove the main result of Gabow et al. [9], namely, deciding if there exists a path in a graph between two given vertices respecting a set of forbidden pairs, is **NP**-complete.

Unfortunately, deciding whether a minor-ancestral graph property is in co-FSP (and therefore determining the complexity of corresponding forbidden pairs problem) appears to be hard. In particular, we now show that deciding membership in co-FSP is **NP**-hard for arbitrary minor-ancestral properties presended as sets of characteristic minors.

3.1 Deciding Membership in co-FSP

Before we proceed we need a few more definitions and lemmas. A *claw* is the graph $K_{1,3}$, i.e., the complete bipartite graph on a bipartition with 1 vertex on one side and 3 vertices on the other. In any graph, edges on dangling paths are said to be *external*; all other edges are *internal*. We say a graph G *minor-covers* a graph H, if for all internal $e \in E_H$, one can subdivide e a finite number of times to produce a graph of which G is a minor.

Lemma 4. *G-minor-containment is equivalent to G-subgraph-containment if and only if G is the disjoint union of paths and subdivisions of claws.*

Proof. Minor-containment and topological-minor-containment are equivalent for graphs with maximum degree 3 [6, p. 21]. Lemma 3.4 of [5] shows topological-minor-containment and subgraph-containment are equivalent only for graphs with no internal edges. The disjoint unions of paths and subdivisions of claws are the only graphs satisfying these conditions. □

Lemma 5. *Let G be a disjoint union of paths and subdivisions of claws, and H be a graph with internal edges and maximum vertex degree 3. The minor-ancestral property Π with $C_\Pi = \{G, H\}$ is in co-FSP if and only if G minor-covers H.*

Proof. Suppose G minor-covers H. We will show that H-minor-containment is equivalent to a finite number of subgraph-containment queries. We claim that the following finite set of graphs serves witness to Π's membership in co-FSP: $\{G, H\} \cup \{$all graphs with an H minor and size bounded by $|E_H|(1+|V_G|)\}$. Let H' be an arbitrary graph larger than anything in the above set with an H-minor. As H has a maximum vertex degree of 3, H-minor- and H-topological-minor-containment are equivalent. Therefore, H' has a smallest subgraph H'' that is a subdivision of H. If $|V_{H''}| \leq |E_H|(|V_G| + 1)$ then H' has a subgraph in the witness set. Otherwise, some edge in H must be subdivided more than $|V_G|$ times to produce H''. Thus, G is a minor of H'' and thus of H'. By Lemma 4, G is a subgraph of H', so the above-listed set contains a subgraph of each graph with an H minor.

On the other hand, suppose that G does not minor-cover H. For contradiction, assume that Π is in co-FSP, i.e., there exists some finite set T such that any graph in Π has a subgraph in T. Let k be the size of the largest graph in T.

As G does not minor-cover H, H has an internal edge uv such that no amount of subdividing uv will result in a graph having G as a minor. Let H' be the graph resulting in subdividing uv k times, and let P be the path from u to v resulting from the subdivision.

H' is in Π, so H' has a subgraph H'' in T. H'' must have an H minor because it does not have G as a minor. As H'' has at most k vertices, it cannot span P. This implies that the edges in H'' isomorphic to edges in P must be external. But now H'' has fewer internal edges than H (as uv was internal), which is a contradiction as contraction and deletion can only reduce the number of internal edges in a graph and H'' was presumed to have an H minor. As no finite set T contains a subgraph of every graph in Π, Π cannot be in co-FSP. □

Theorem 2. *Given C_Π, deciding if a nontrivial minor-ancestral property Π is in co-FSP is* ***NP****-hard.*

Proof. We reduce from the Hamiltonian Path problem, well known to be **NP**-complete, even for graphs with maximum degree 3 [10]. Let G be an arbitrary graph with maximum vertex degree 3 to act as input for Hamiltonian Path. We can immediately dismiss the case where G has no internal edges as clearly this can be decided in polynomial time. Assume that G has an even number of vertices (if not, just subdivide an edge in it).

Let H be the subdivision of a claw having three dangling paths of lengths $|V_G|/2$, $|V_G|/2$, and $|V_G|$, respectively. Assign an arbitrary order to the edges of G. Make $|E_G|$ copies of G, one for each edge of G, labelled $G_1, ..., G_{|E_G|}$. For each G_i, subdivide the edge corresponding to edge i in G_i and attach a dangling path P_i of length $|V_G|$ to vertex created during subdivision. Finally, attach a vertex to the penultimate vertex in the dangling path, so the addition consists of two external and $|V_G| - 1$ internal edges. Let G' be the disjoint union of G_1 to $G_{|E_G|}$. Notice that as G's vertices each have degree at most 3, the same is true of G'. Define the minor-ancestral graph property Π by letting $C_\Pi = \{G', H\}$.

Suppose that G contains a Hamiltonian path. As $|V_G|$ is even, there is a unique middle edge in the path with index i. H is a subgraph of G_i and thus of G'. Therefore H minor covers G', proving Π is in co-FSP.

In the other direction, suppose that H minor-covers G' (so, by Lemma 5, Π is in co-FSP). Note that by Lemma 4, H-minor- and H-subgraph-containment are equivalent. Clearly if H is a subgraph of some G_i then G has a Hamiltonian path as each G_i has one more vertex than H and one of the leaves of P_i cannot be covered by H.

No matter how many times we subdivide internal edges on some P_i, as the longest dangling path in H has $|V_G|$ vertices, we can only use that many to demonstrate H is a subgraph of G_i. Furthermore, one end of P_i consists of just two vertices, so one must be able to identify the remaining vertices of H (excluding the dangling path identified to vertices of P_i) to all vertices of G. Therefore, H must be a subgraph of G' without the subdivision, which implies G has a Hamiltonian path.

Thus we have, G contains a Hamiltonian path if and only if H minor-covers G' and by Lemma 5 this occurs if and only if Π is in co-FSP. □

3.2 Forbidden Edge Pairs

As an extension of our main result we consider the variant of the problem with forbidden edge pair constraints. Given a graph property Π, let $\text{FEP}_\Pi(G, F)$ be true if and only if there exists a G' a subgraph of G such that $E_{G'} \times E_{G'}$ and F are disjoint where F is a set of edge pairs of G. We have the following result corresponding to Theorem 1.

Theorem 3. *If minor-ancestral property Π is in co-FSP then* FEP_Π *is in* ***P****, otherwise* FEP_Π *is* ***NP****-complete.*

Proof. Clearly Lemma 2 holds by just checking edges instead of vertices for violations of the forbidden pairs. Lemma 3 follows by replacing each $v_{i,j}$ with two vertices connected by an edge and forbidding the corresponding edges. □

3.3 Fixed Parameter Tractability

Finally we observe that all of the problems shown **NP**-complete by our main result are all tractable when the size of the forbidden vertex pairs set is fixed. A problem is *fixed parameter tractable* [7] in a parameter k if there is an algorithm that runs in time $f(k) * n^{O(1)}$, where f is a function independent of the size of the input problem, n. In other words, holding the parameter k constant means that the complexity of the problem grows polynomially with the remainder of the input.

Theorem 4. *Let Π be a minor-ancestral graph property. Then* FVP_Π *is fixed parameter tractable on the number of forbidden vertex pairs.*

Proof. Suppose that an input to FVP_Π has no more than k forbidden pairs. From the k pairs, we can derive at most 2^k different ways to pick one vertex from each pair. For each way, delete the vertices not picked and check the resulting graph for the property. If one of these has it, then answer affirmatively, otherwise reject. This algorithms requires at most a constant 2^k calls to a polynomial-time check (since Π is minor-ancestral) for the presence of a subgraph with a property. □

Note that the above theorem applies to any property that can be decided in polynomial time, not just minor-ancestral properties as the proof only used the fact that the decision property for minor-ancestral properties is in **P**.

4 Conclusions

We note that Lemma 2 and the technique used in Lemma 3 can be applied to graph properties that are not minor-ancestral. For example, deciding if a graph contains a triangle is not minor-ancestral but Lemma 2 can be used to show that deciding if a graph has a triangle respecting a set of forbidden vertex pairs is in **P**. Perhaps more interesting is the existence of non-minor-ancestral decision problems that are in **P** without constraints but become **NP**-complete when forbidden vertex pairs are added. For example, consider the property satisfied by graphs that contain a perfect matching. Using a construction similar to that used in Lemma 3 one can show that the problem of deciding if a graph has a perfect matching respecting a given set of forbidden edge pairs is **NP**-complete. We further note that for any nontrivial subgraph-hereditary property the corresponding problem with forbidden vertex pairs is trivially in **P** as the empty graph respects any set of constraints on the vertices chosen. This leaves open the question of characterizing exactly which easy graph properties become hard when forbidden vertex pairs constraints are considered.

Acknowledgments. We thank our anonymous reviewers and Ben Hescott for invaluable, thorough feedback and edits. Much gratitude goes to Robin Kothari for particularly insightful discussions.

References

1. Alon, N., Shapira, A., Sudakov, B.: Additive approximation for edge-deletion problems. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2005, pp. 419–428. IEEE Computer Society, Washington, DC (2005)
2. Asano, T., Hirata, T.: Edge-deletion and edge-contraction problems. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC 1982, pp. 245–254. ACM, New York (1982)
3. Bondy, J.A.: Graph Theory With Applications. Elsevier Science Ltd. (1976)
4. Brady, A., Maxwell, K., Daniels, N., Cowen, L.J.: Fault tolerance in protein interaction networks: Stable bipartite subgraphs and redundant pathways. PLoS One 4(4), e5364 (2009)
5. Childs, A.M., Kothari, R.: Quantum query complexity of minor-closed graph properties. In: Proc. 28th Symposium on Theoretical Aspects of Computer Science (STACS 2011). Leibniz International Proceedings in Informatics, vol. 9, pp. 661–672 (2010)
6. Diestel, R.: Graph Theory. Springer, Heidelberg (2010)
7. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
8. Fellows, M.R., Langston, M.A.: Nonconstructive tools for proving polynomial-time decidability. J. ACM 35, 727–739 (1988)
9. Gabow, H.N., Maheshwari, S.N., Osterweil, L.J.: On Two Problems in the Generation of Program Test Paths. IEEE Transactions on Software Engineering SE-2(3), 227–231 (1976)
10. Garey, M.R., Johnson, D.S., Endre Tarjan, R.: The Planar Hamiltonian Circuit Problem is NP-Complete. SIAM Journal on Computing 5(4), 704–714 (1976)
11. Kolman, P., Pangrác, O.: On the complexity of paths avoiding forbidden pairs. Discrete Applied Mathematics 157(13), 2871–2876 (2009)
12. Krause, K.W., Goodwin, M.A., Smith, R.W.: Optimal software test planning through automated network analysis. TRW Systems Group, Redondo Beach (1973)
13. Krishnamoorthy, M.S., Deo, N.: Node-Deletion NP-Complete Problems. SIAM Journal on Computing 8(4), 619–625 (1979)
14. Lewis, J., Yannakakis, M.: The node-deletion problem for hereditary properties is NP-complete. Journal of Computer and System Sciences 20(2), 219–230 (1980)
15. Robertson, N., Seymour, P.D.: Graph Minors. I. Excluding a Forest. Journal of Combinatorial Theory, Series B 35(1), 39–61 (1983)
16. Robertson, N., Seymour, P.D.: Graph Minors. XIII. The Disjoint Paths Problem. Journal of Combinatorial Theory, Series B 63(1), 65–110 (1995)
17. 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)
18. Yinnone, H.: On paths avoiding forbidden pairs of vertices in a graph. Discrete Appl. Math. 74, 85–92 (1997)

Satisfiability Thresholds beyond $k-$XORSAT

Andreas Goerdt and Lutz Falke

Technische Universität Chemnitz, Fakultät für Informatik
Straße der Nationen 62, 09107 Chemnitz, Germany
{goerdt,falu}@informatik.tu-chemnitz.de
http://www.tu-chemnitz.de/informatik/TI/

Abstract. We consider random systems of equations $x_1 + \cdots + x_k = a$, $0 \leq a \leq 2$ which are interpreted as equations modulo 3. We show for $k \geq 15$ that the satisfiability threshold of such systems occurs where the $2-$core has density 1. We show a similar result for random uniquely extendible constraints over 4 elements. Our results extend previous results of Dubois/Mandler for equations mod 2 and $k = 3$ and Connamacher/Molloy for uniquely extendible constraints over a domain of 4 elements with $k = 3$ arguments.

The proof is based on variance calculations, using a technique introduced by Dubois/Mandler. However, several additional observations (of independent interest) are necessary.

1 Introduction

1.1 Contribution

Often constraints are equations of the type $f(x_1, \ldots, x_k) = a$ where a is an element of the domain considered and f is a $k-$ary function on this domain, for example addition of k elements. Given a formula, which is a conjunction of m constraints over n variables we want to find a solution. It is natural to assume that f has the property: Given $k-1$ arguments we can always set the last argument such, that the constraint becomes true. In this case we can restrict attention to the $2-$core. It is obtained by iteratively deleting all variables which occur at most once. Thus it is the maximal subformula in which each variable occurs at least twice.

We consider the random instance $F(n,p)$: Each equation over n variables is picked independently with probability p; the domain size d and the number of slots per equation k is fixed. We consider the case $p = c/n^{k-1}$, and the number of constraints is linear in n whp. (with high probability, that is probability $1 - o(1), n$ large.) Note that the number of equations available is $n^k \cdot d$. The density of a formula is equal to the number of equations divided by the number of variables. The following is well known for $F(n,p)$ and $k \geq 3$ which we assume throughout.

Fact 1 ([22]). *1. Conditional on the number of variables n' and equations m' of the $2-$core the $2-$core is a uniform random member of all formulas where each variable occurs at least twice.*

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 148–159, 2012.

2. There exist $n' = n'(c)$ and $m' = m'(c)$ such that the number of variables of the $2-$core is $n'(1+o(1))$ and the number of equations $m'(1+o(1))$ whp.
3. There exists a $T = T(k)$ such that whp. for $c \le T - \varepsilon$ the $2-$core has density $\le 1-\varepsilon'$ and for $c \ge T + \varepsilon$ the $2-$core has density $\ge 1+\varepsilon'$.

For any formula with density ≥ 1 the density of the $2-$core is ≥ 1 as the density does not decrease when a variable of degree ≤ 1 is deleted. Thus T is such that the density of $F(n,p)$ is ≤ 1. Let b be the density of $F(n,p)$ and let $B = B(T)$ be the threshold density. For $k = 3,4,5$ we have $B = 0.9179\ldots,\ 0.976\ldots,\ 0.9924\ldots$ [11]. An analytical determination of B following [11]: Consider $F(\beta) = \beta/k \cdot 1\,/\,\Pr[\mathrm{Po}(\beta) \ge 1]^{k-1}$. $\mathrm{Po}(\beta)$ is the Poisson distribution with parameter $\beta > 0$, $\Pr[\mathrm{Po}(\beta) \ge 1] = 1 - \exp(-\beta)$. $F(\beta) \ge 0$, convex, and has a unique minimum. The minimum of $F(\beta)$ is the threshold density for $F(n,p)$ to have a non-empty $2-$core. For $b >$ this threshold density let $\beta(b)$ be uniquely determined as the larger solution of $F(\beta(b)) = b$. Consider $G(\beta) = \beta\,/\,k \cdot \Pr[\mathrm{Po}(\beta) \ge 1]/\Pr[\mathrm{Po}(\beta) \ge 2]$. The unique b with $G(\beta(b)) = 1$ is the threshold density $B = B(T)$.

The expected number of solutions of the $2-$core is $d^{n'-m'}$. When the $2-$core has density $\ge 1+\varepsilon$ whp. no solution exists. In seminal work Dubois and Mandler [12] consider equations $\mod 2 : x_1 + \ldots + x_k = a$, $0 \le a \le 1, k = 3$. They show satisfiability whp. when the 2-core has density $\le 1-\varepsilon$. For larger $k \ge 15$ a full proof for this result is given in [11], Appendix C. Thus T/n^{k-1} is the threshold for unsatisfiability in this case.

It is a natural conjecture that the same threshold applies to equations as discussed initially (and to similar types.) However, it seems difficult to prove the conjecture in some generality. Considerable effort in this direction has been spent, see [8] described further below. One of the difficulties seems to be that we have 2 parameters k and d. We make some progress towards this conjecture. We show it for equations $\mod 3$. (The result is for $k > 15$, but we think one can get if for all $k \ge 3$ along the lines presented.)

Theorem 2. *Let $F(n,p)$ be the random set of equations $\mod 3 : x_1 + \cdots + x_k = a$, $0 \le a \le 2$, $x_1 + \cdots + x_k$ an ordered $k-$tuple of variables, $k > 15$. If $p < (T-\varepsilon)/n^{k-1}$ $F(n,p)$ is satisfiable whp.*

The main task is to show that a $2-$core of density $\le 1-\varepsilon$ has a solution with probability $> \varepsilon' > 0$. Our proof starts as Dubois/Mandler: Let X be the number of satisfying assignments of the $2-$core. Its expectation is $\ge d^{\varepsilon n}, d = 3$. We show that $E[X^2] \le O(E[X])^2$. This implies (by Cauchy-Schwartz (or Paley-Zygmund) inequality) that the probability to have a solution is $\ge \varepsilon' > 0$. Fact 1 implies that $F(n,p)$ has a solution with the same probability. We apply Friedgut-Bourgain's Theorem to $F(n,p)$ to show that unsatisfiability has a sharp threshold. By this the probability becomes $1 - o(1)$. In [9] Friedgut–Bourgain is applied to the $\mod 2-$case. It seems that our proof for the $\mod 3-$case is somewhat simpler (and applies to the $\mod 2-$case and other cases.)

To determine $E[X^2]$ Dubois/Mandler apply Laplace Method (one ingredient: bounding a sum through its maximum term.) The main difficulty is to bound a real valued function of several arguments from above. They show that their

function has only one local maximum. We proceed by the same method, but substantial changes are necessary for $k > 3$.

First, we observe (cf. [11], Appendix C) that the function in question is $\leq$ the *infimum* with respect to certain other parameters. This is based on generating functions: If $f(x) = \sum c_k x^k$ then $c_k \leq f(a)/a^k$, $a > 0, c_i \geq 0$ (a method rarely used in the area, a notable exception is [24].) Thus to bound the maximum from above we need to find suitable parameters and show that the value of the function with these parameters is less than the required upper bound. (This leads to involved, but elementary calculus.)

To make this approach work we need appropriate generating functions: $X = X_{a_1} + \cdots + X_{a_{3^n}}$, where X_{a_i} is the indicator random variable of the event that assignment a_i makes the formula true. Then $X^2 = \sum_a \sum_b X_a X_b$. To get $E[X^2]$ we need to determine $\text{Prob}[X_a X_b = 1]$. To this end we observe that the equation $x_1 + \cdots + x_k = c$ which is true under a is true exactly under those assignments b such that $0k_0 + 1k_1 + 2k_2 = 0 \mod 3$, and k_i is the number of slots of $x_1 + \cdots + x_k$ filled with a variable x with $b(x) = a(x) + i$. Thus there are $\sum_{k_1 = k_2 \mod 3} \binom{k}{k-k_1-k_2, k_1, k_2}$ different ways in which an equation can become true under a, b. The following generating function allows us to deal with these possibilities analytically. With $\mathbf{w_1} = \exp(\mathbf{2}\pi\imath/\mathbf{3})$ the primitive third root of unity and $\mathbf{w_2} = \mathbf{w_1^2}$ we define $r(x_0, x_1, x_2) =$
$1/3 \cdot \left[(x_0 + x_1 + x_2)^k + (x_0 + \mathbf{w_1} x_1 + \mathbf{w_2} x_2)^k + (x_0 + \mathbf{w_2} x_1 + \mathbf{w_1} x_2)^k\right]$
then $\text{Coeff}[x_1^{k_1} x_2^{k_2}, r(1, x_1, x_2)] = \binom{k}{k-k_1-k_2, k_1, k_2}$ if $k_1 = k_2 \mod 3$ and 0 otherwise (easy from properties $\mathbf{w_j}$.) In the $\mod 2$−case $1/2\left[(1+x)^k + (1-x)^k\right]$ serves the same purpose.

With the motivation to get an exact threshold of unsatisfiability for a type of constraint whose worst-case complexity is NP-complete, Connamacher/Molloy [8], see also the very recent [7] introduce uniquely extendible constraints. A k−ary uniquely extendible constraint is a function from D^k to true, false with the property: Given values from D for any $k-1$ argument slots there is exactly one value for the remaining slot which makes the constraint true. Note that equations as above are uniquely extendible, but the class of all uniquely extendible constraints is larger. (The $k > 8$ in the following result can be eliminated at the price of some additional technical effort.)

Theorem 3. *Let $F(n,p)$ be the random formula of uniquely extendible constraints: Each constraint is a random k−tuple of variables and a random k−ary uniquely extendible constraint over D and we pick with probability p and $k > 8$. For $|D| = 4$ and $p < (T-\varepsilon)/n^{k-1}$ $F(n,p)$ is satisfiable whp.*

The threshold T/n^{k-1} is proved for $k = 3$ and $|D| = 4$, cf. [7] remark following Theorem 8. Our proof uses the technique as in the $\mod 3$−case, however the details are different. One of the contributions making the proof possible is the generating polynomial $p(x) = \frac{1}{d}\left[(1+x)^k + (d-1)(1 - \frac{x}{d-1})^k\right]$ (observe $d = 2$.)

In the main part we give the structure of the proof of Theorem 2. Concerning Theorem 3 some characteristic points are given. Full proofs in [16].

1.2 Motivation

Many computational problems can be naturally formulated as conjunctions of constraints. And we are interested to find a solution of this conjunction. Algorithmic properties of these conjunctions are considered in theoretical research (with remarkable results e. g. in the realm of approximation [17]) and applied research, e. g. [20]. An additional aspect is the investigation of conjunctions of randomly picked constraints; [23] is a fundamental study here. Propositional formulas in k-conjunctive normalform provide an example which has lead to a rich literature e. g. [10]. One of the characteristic properties of this research is that its findings can often be related to experimental work by running algorithms on randomly generated instances. A point of interest of random formulas is the threshold phenomenon fur satisfiability. Typically, instances picked close to the threshold seem to be algorithmically hard, thus being candidate test cases for algorithms. The threshold phenomenon and the possibility to investigate it by experiments is one of the reasons that physics is interested in the area e. g. [4]. On the other hand, physical approaches lead to new algorithms and classical theoretical computer science research, e. g. [6].

One of the major topics is to determine the value of the threshold. A full solution even in the natural $k-$CNF SAT case, $k \geq 3$, has not been obtained, but many partial results, [10] for $k = 3$. Note that $k-$CNF does not have the unique extendability property as possessed by the constraints considered here. And it seems to be a major open problem to get the precise threshold for constraints without unique extendability and not similar to $2-$CNF. A mere existence result is the Friedgut-Bourgain theorem [14]. Based on this theorem thresholds for formulas of constraints over domains with more than 2 elements are considered in [23]. Ordering constraints are considered in [15], only partial results towards a threshold can be proven. In order to get definite threshold results further techniques are required. Therefore it is a useful effort to further develop the techniques with which thresholds can be proven. This is the general contribution of this paper.

A notable early exception, in that the precise threshold can be proven is the mod $2-$case considered above. Historically [12] is the first paper which uses variance calculation based on Laplace method in this area. Subsequently, for $k-$CNF SAT this method has lead to substantial progress in [1]. The contribution here is that mod $2-$proof can be refined and extended to cover other cases based on observations of independent interest. Note that random sparse linear systems over finite fields are used to construct error correcting codes, e. g. [19] or [25], motivating the mod $3-$case; literature on ternary linear codes can also be found. A very recent study of the mod $2-$case is [2]. Related literature can be found in [18], but precise threshold results have not been obtained here.

2 Notation and Basics for the mod 3−Case

We use the abbreviation

$$M(m,n) := \sum_{v_1,\dots v_n \geq 2} \binom{m}{v_1,\dots v_n} \text{ and } N_0 := M(km,n). \text{ Then } N_0 \cdot 3^m \quad (1)$$

is the number of all formulas with k variables per equation and m equations. Note that N_0 is the number of ways to partition km slots such that we have ≥ 2 slots for each of the n variables. The formulas we consider are 2−cores. (Here the same equation can occur several times. This happens with probability $o(1)$ as m is linear in n and can be ignored.) We consider the uniform distribution on the set of formulas. Let X be the number of solutions of a formula. We have $X = \sum_a X_a$ where a stands for an assignment of the variables with $0, 1, 2$ and $X_a(F) = 1$ if F is true under a and 0 otherwise. The expectation of X is 3^{n-m} because given an assignment each equation is true independently with probability $1/3$. We assume that $m = \gamma n, \gamma$ bounded above by a constant < 1. As k is constant, the asymptotics is only with respect to n. We need to show the following theorem:

Theorem 4. $E[X^2] \leq C \cdot 3^{2(n-m)}$.

We have $\mathrm{E}[X^2] = \sum_a \sum_b \mathrm{E}[X_a \cdot X_b]$. Let $\overline{W} = (W_0, W_1, W_2)$ be a partition of the set of variables into 3 sets. We always use the notation $w_i = \sharp W_i$, $\bar{w} = (w_0, w_1, w_2)$. For two assignments a, b we write $b = D(a, \overline{W})$ iff $W_i = \{x \mid b(x) = a(x) + i \mod 3\}$. We have that $a(x_1 + \cdots + x_k) = b(x_1 + \cdots + x_k)$ ($a(x_1 + \cdots + x_k)$ is the value of $x_1 + \cdots + x_k$ under a, analogously for b) iff $\sum_{i=0,1,2} i \cdot \sharp\{j \mid x_j \in W_i\} = 0 \mod 3$. This is equivalent to $\sharp\{j \mid x_j \in W_1\} = \sharp\{j \mid x_j \in W_2\} \mod 3$. Given $\bar{l} = (l_0, l_1, l_2)$ with $\sum l_i = km$ we let $\mathcal{K}(\bar{l})$ be the set of all $3 \times m$−matrices $(k_{i,j})_{0 \leq i \leq 2, 1 \leq j \leq m}$ with $k_{1,j} = k_{2,j} \mod 3$ and each column sums to k, that is $\sum_i k_{i,j} = k$ for each j. Moreover, $\sum_j k_{i,j} = l_i$ for $i = 0, 1, 2$ (the i'th row sums to l_i.) We denote $K(\bar{l}) :=$

$$\sum_{(k_{i,j}) \in \mathcal{K}(\bar{l})} \prod_{j=1}^{m} \binom{k}{k_{0,j}, k_{1,j}, k_{2,j}}. \text{ Then } \hat{N}(\bar{w}, \bar{l}) := K(\bar{l}) \cdot \prod_{i=0}^{2} M(l_i, w_i) \quad (2)$$

is the number of formulas F true under two assignments a, b with $b = D(a, \overline{W})$ (with $w_i = \sharp W_i$) and the variables from W_i occupy exactly l_i slots of F. The factor $K(\bar{l})$ of $\hat{N}(\bar{w}, \bar{l})$ counts how the l_i slots available for W_i are distributed over the left-hand-sides of the equations. The second factor counts how to place the variables into their slots. Note that the right-hand-side of an equation cannot be chosen, it is determined by the value of the left-hand-side under a, b.

We abbreviate $\binom{n}{\bar{w}} = \binom{n}{w_0, w_1, w_2}$. Given an assignment a, $\bar{w}$, and $\bar{l}$, the number of assignment formula pairs (b, F) with : There exist $\overline{W}$ with $\sharp W_i = w_i$, such that $b = D(a, \overline{W})$, F is true under a and b, and the variables from W_i occupy

exactly l_i slots of F is

$$N(\bar{w},\bar{l}) := \binom{n}{\bar{w}} \cdot \hat{N}(\bar{w},\bar{l}). \text{ Then } \mathrm{E}[X^2] = 3^n \cdot \sum_{\bar{w},\bar{l}} N(\bar{w},\bar{l}) \cdot \frac{1}{3^m \cdot N_0}. \quad (3)$$

Theorem 4 follows directly from the next theorem:

Theorem 5. $\sum_{\bar{w},\bar{l}} N(\bar{w},\bar{l})/N_0 \le C \cdot 3^{(1-\gamma)n}$.

One more piece of notation: $\omega_i = w_i/n$ is the fraction of variables belonging to W_i. And $\lambda_i = l_i/(km) = l_i/(k\gamma n)$ is the fraction of slots filled with a variable from W_i, $\omega_0 = 1-\omega_1-\omega_2$ (also for λ_0.) We use $\bar{\omega} = (\omega_0,\omega_1,\omega_2)$, and $\bar{\lambda} = (\lambda_0,\lambda_1,\lambda_2)$. Sometimes ω_i, λ_i stand for arbitrary reals.

3 Structure of the Proof of Theorem 5

First, bounds for $M(m,n)$ and $K(\bar{l})$. We consider $q(x) := \exp(x) - x - 1 = \sum_{j\ge 2} \frac{x^j}{j!}$ for $x \ge 0$. Then for $a > 0$ and all m, n we have $M(m,n) =$

$$= \mathrm{Coeff}[x^m, q(x)^n] \cdot m! < q(a)^n \cdot \frac{1}{a^m} \cdot m! \le q(a)^n \left(\frac{m}{a\cdot e}\right)^m \cdot O(\sqrt{m}) \quad (4)$$

using Stirling in the form $m! < (m/e)^m \cdot O(\sqrt{m})$. To get rid the $\sqrt{m}-$factor we let $Q(x) := xq'(x)/q(x)$ with $q'(x)$ the derivative of $q(x)$, $q'(x) = \exp(x) - 1$ for $x > 0$. Observe that $Q(x)$ is the expectation of a random variable with probability of $j \ge 2$ equal to $(x^j/j!)/q(x)$. Then $Q'(x) > 0$ for $x > 0$, $Q(x) > x$, and $Q(x) \longrightarrow 2$ for $x \longrightarrow 0$. Thus, for $y > 2$ the inverse function $Q^{-1}(y) > 0$ is defined and differentiable. Lemma 6 follows from the local limit theorem for sums of lattice random variables [13], page 112 which generalizes the fact that $\binom{n}{n/2} \cdot 1/2^n = \Theta(1/\sqrt{n})$ (by Stirling.) The $\sqrt{m}-$factor cancels.

Lemma 6 *Let $Cn \ge m \ge (2+\varepsilon)n$, $C, \varepsilon > 0$ constants. Then*

$$M(m,n) = \Theta(1) \cdot \left(\frac{m}{ae}\right)^m \cdot q(a)^n \text{ with } a \text{ defined by } Q(a) = \frac{m}{n}.$$

Throughout we use $s = s(k,\gamma)$ uniquely defined by $Q(s) = k\gamma = k\gamma n/n = km/n$, the average number of occurrences of a variable. For $k \ge 3$ we can assume that $k\gamma > 2$ and s always exists. We have $Q := Q(s) \ge s$. Recall $N_0 = M(km,n)$ and we get a tight bound on the number of formulas (cf. (1).)

Corollary 7. $N_0 = \Theta(1)\,(k\gamma n/(se))^{k\gamma n} \cdot q(s)^n$.

We treat the sum $K(\bar{l})$ analogously to $M(m,n)$. Instead of $q(x)$ we use the function,

$$r(\bar{x}) := \sum_{k_1=k_2 \bmod 3} \binom{k}{k_0,k_1,k_2} x_0^{k_0} x_1^{k_1} x_2^{k_2},\ \bar{x} = (x_0,x_1,x_2). \text{ Then}$$

$$K(\bar{l}) = \sum_{(k_{i,j})\in\mathcal{K}(\bar{l})} \prod_{j=1}^{m} \binom{k}{k_{0,j}, k_{1,j}, k_{2,j}} = \mathrm{Coeff}[\bar{x}^{\bar{l}},\, r(\bar{x})^m\,] < \frac{r(\bar{c})^m}{\bar{c}^{\bar{l}}} \quad (5)$$

with the notation $\bar{x}^{\bar{l}} = \prod_i x_i^{l_i}$ and $\bar{c} = (c_0, c_1, c_2) > 0$, meaning $c_i > 0$ for all i.

For calculations it is useful to have a different representation of $r(\bar{x})$. Let $\imath$ be the imaginary unit, and $\mathbf{w_1} := -1/2 + (\sqrt{3}/2)\imath$ is the primitive third root of unity, $\mathbf{w_2} := -1/2 - (\sqrt{3}/2)\imath = \mathbf{w_1}^2$. We have $r(\bar{x}) =$

$$\frac{1}{3}\left[(x_0 + x_1 + x_2)^k + (x_0 + \mathbf{w_1}x_1 + \mathbf{w_2}x_2)^k + (x_0 + \mathbf{w_2}x_1 + \mathbf{w_1}x_2)^k\right] \quad (6)$$

Note that in derivatives $\frac{d}{dx_i} r(\bar{x})$ the roots of unity are treated as constants.

For $x_i, y_i > 0$ we define (convention $\alpha^\alpha = 1$ for $\alpha = \omega_i$ or $\alpha = \lambda_i$ and $\alpha = 0$)

$$\Psi(\bar{\omega}\,,\,\bar{\lambda}\,,\,\bar{x}\,,\,\bar{y}\,) = \prod_{i=0,1,2}\left(\frac{q(x_i)}{\omega_i q(s)}\right)^{\omega_i} \cdot \left[\prod_{i=0,1,2}\left(\frac{\lambda_i s}{x_i y_i}\right)^{\lambda_i}\right]^{k\gamma} r(y_0, y_1, y_2)^\gamma$$

With $\omega_i = \lambda_i = 1/3, a_i = s(k,\gamma) = s$, and $c_i = 1$, we have $\Psi(\overline{\omega}, \overline{\lambda}, \bar{a}, \bar{c}) = 3 \cdot (1/3)^{k\gamma} \cdot ((1/3)3^k)^\gamma = 3^{1-\gamma}$ (use (6).) The next lemma shows how Ψ bounds the exponential part of $N(\bar{w}, \bar{l})/N_0$.

Lemma 8 *$N(\bar{w}, \bar{l})/N_0 < \Psi(\bar{\omega}\,,\,\bar{\lambda}\,,\,\bar{a}\,,\,\bar{c}\,)^n \cdot O(n)^{3/2}$ for any $\bar{a}, \bar{c} > 0$.*

Proof. $\binom{n}{\bar{w}} \le \prod_i (1/\omega_i)^{\omega_i n}$ for all $\bar{w}$ ([21], page 228) $\prod_{i=0,1,2} M(l_i, w_i)/N_0 \le \prod_i \left((l_i/(a_i e))^{l_i} q(a_i)^{w_i} O(\sqrt{l_i})\right) \cdot (es/(k\gamma n))^{k\gamma n} \cdot 1/q(s)^n \cdot O(1)$ with (4) and Corollary 7. Observe that $l_i = \lambda_i k\gamma n, \sum_i \lambda_i = 1, \sum \omega_i = 1$. Concerning $K(\bar{l})$ apply (5).

For reals a, b we let $\mathcal{U}_\varepsilon(a,b) = \{(c,d)|\ |c-a|, |d-b| < \varepsilon\}$ be the open square neighborhood of (a,b). The notation $\bar{\lambda}, \bar{\omega} \in \mathcal{U}_\varepsilon(a,b)$ is used to mean $(\lambda_1, \lambda_2), (\omega_1, \omega_2) \in \mathcal{U}_\varepsilon(a,b)$. The proof of Theorem 9 is outlined in Section 4 which is the technical contribution.

Theorem 9. *For any $\bar{\omega},\ \bar{\lambda} > 0$ there exist $\bar{a}, \bar{c} > 0$ such that:*
(1) $\Psi(\bar{\omega}\,,\,\bar{\lambda}\,,\,\bar{a}\,,\,\bar{c}\,) \le 3^{1-\gamma}$.
(2) For any $\varepsilon > 0$, if $\bar{\lambda} \notin \mathcal{U}_\varepsilon(1/3, 1/3)$ then $\Psi(\bar{\omega}\,,\bar{\lambda}\,,\bar{a}\,,\bar{c}) \le 3^{1-\gamma} - \delta$ for a $\delta > 0$.

To prove Theorem 5 we split the sum depending on the value of $\bar{\lambda}$. First the case that $\bar{\lambda}$ is not close to $(1/3, 1/3, 1/3)$.

Corollary 10. *Let $U = \mathcal{U}_\varepsilon(1/3, 1/3)$ then $\sum_{\bar{\lambda} \notin U, \lambda_i > 0, \bar{\omega}} N(\bar{w}, \bar{l})/N_0 < C \cdot 3^{(1-\gamma)n}$.*

Proof. The sum has only $O(n^4)$ terms. With Lemma 8 and Theorem 9 (2) we see that each term is bounded above by $(3^{1-\gamma} - \delta)^n O(n)^{3/2}$.

To treat $\bar{\lambda}$ close to $(1/3, 1/3, 1/3)$ we need a better upper bound for Ψ. Therefore we need a lemma analogous to Lemma 6 for $K(\bar{l})$. Let the function $R(x_1, x_2) = (R_1(x_1, x_2),\ R_2(x_1, x_2))$ be defined by $R_i(x_1, x_2) =$
$= x_i r_{x_i}(1, x_1, x_2)/r(1, x_1, x_2)$ for $i = 1, 2$, $r_{x_i}(1, x_1, x_2)$ is the partial derivative of $r(1, x_1, x_2)$ wrt. x_i. Observe that $R(x_1, x_2)$ is the expectation of the random

vector with probability of (j_1, j_2) with $j_1 = j_2 \mod 3$ equal to $\binom{k}{k-j_1-j_2, j_1, j_2} \cdot x_1^{j_1} x_2^{j_2} / r(1, x_1, x_2)$. The Jacobi Determinant of $R(x_1, x_2)$ is > 0 at $x_1 = x_2 = 1$. Note that we consider only the neighborhood of $(x_1, x_2) = (1, 1)$. This makes the calculation of the determinant possible. Thus there is a neighborhood of $(1, 1)$ in which $R(x_1, x_2)$ is invertible and the inverse function is differentiable. We have that $R(1, 1) = (k/3, k/3)$. Thus for a suitable ε and $(\lambda_1, \lambda_2) \in \mathcal{U}_\varepsilon(1/3, 1/3)$ we can define (c_1, c_2) by $R(c_1, c_2) = (k\lambda_1, k\lambda_2)$. Moreover, $c_i = c_i(\lambda_1, \lambda_2)$ is differentiable. Lemma 11 follows from a local limit theorem for sums of $2-$dimensional lattice-valued random vectors [3], page 231 ff. It is a generalization of the fact that $\binom{n}{n/3, n/3, n/3} \cdot 1/3^n = \Theta(1) \cdot 1/n$ from Stirling. (Consider the 2-dimensional random vector $(X_1, X_2) = (0, 0), (0, 1), (1, 0)$ each with probability $1/3$.)

Lemma 11 *There is an $\varepsilon > 0$ such that for $(\lambda_1, \lambda_2) \in \mathcal{U}_\varepsilon(1/3, 1/3)$*

$$K(\bar{l}) = O\left(\frac{1}{n}\right) \cdot \frac{r(1, c_1, c_2)}{c_1^{l_1} c_2^{l_2}} \text{ with } R(c_1, c_2) = (k\lambda_1, k\lambda_2) \text{ defining } c_1, c_2.$$

By a proof similar to Lemma 8 – one factor $1/n$ comes from the preceding lemma, the second factor $1/n$ comes from Stirling applied to $\binom{n}{\bar{\omega} n}$ – we get the next corollary.

Corollary 12. *There is $\varepsilon > 0$ such that for $(\omega_1, \omega_2), (\lambda_1, \lambda_2) \in \mathcal{U}_\varepsilon(1/3, 1/3)$*

$$\frac{N(\bar{w}, \bar{l})}{N_0} \leq O\left(\frac{1}{n^2}\right) \Psi(\bar{\omega}\,, \bar{\lambda}\,, \bar{a}\,, \bar{c}\,)^n$$

where $Q(a_i) = l_i / w_i = \lambda_i k\gamma / \omega_i$ and $c_0 = 1$ and $R(c_1, c_2) = (\lambda_1 k, \lambda_2 k)$.

Comment. Observe that $\lambda_i k\gamma / \omega_i \approx k\gamma$, $a_i \approx s, c_i \approx 1$.

Lemma 13 *The function $\Psi(\bar{\omega}\,, \bar{\lambda}) = \Psi(\bar{\omega}\,, \bar{\lambda}\,, \bar{a}\,, \bar{c}\,)$ with a_i, c_i given by $Q(a_i) = \lambda_i k\gamma / \omega_i$ and $c_0 = 1$ and $R(c_1, c_2) = (\lambda_1 k, \lambda_2 k)$ has a local maximum with value $3^{1-\gamma}$ for $\lambda_i = \omega_i = 1/3$. In this case we have $a_i = s$ and $c_i = 1$.*

Proof. Standard calculus applied to $\ln \Psi(\bar{\omega}\,, \bar{\lambda})$ with $\omega_1, \omega_2, \lambda_1, \lambda_2$ considered as free variables ($\omega_0 = 1 - \omega_1 - \omega_2$.) Using the definition of a_i, $Q(a_i) = k\gamma\lambda_i / \omega_i)$ the derivatives of a_i cancel in the subsequent calculation. The same applies to c_i. The derivatives are 0 for $\bar{\lambda} = \bar{\omega} = \overline{1/3}$.

$$\frac{d \ln \Psi(\bar{\omega}, \bar{\lambda})}{d\omega_i} = \ln \omega_0 - \ln \omega_i + \ln q(a_i) - \ln q(a_0)$$
$$\frac{d \ln \Psi(\bar{\omega}, \bar{\lambda})}{d\lambda_i} = k\gamma(\ln \lambda_i - \ln \lambda_0 + \ln a_0 - \ln a_i - \ln c_i).$$

It can be shown that the Hessian matrix of $\ln \Psi(\bar{\omega}, \bar{\lambda})$ is negative definite for $\bar{\lambda} = \bar{\omega} = \overline{1/3}$. This implies the lemma.

Previous applications of the technique used here show that the function like $\Psi(\bar{\omega}, \bar{\lambda})$ has only one local maximum in $\bar{\omega}, \bar{\lambda}$. Among others this approach would require in our case to consider the Jacobian determinant of $R(x_1, x_2)$ not only for $x_1 = x_2 = 1$. This seems not easy. Here we consider $\Psi(\bar{\omega}, \bar{\lambda})$ only in the neighborhood of $\lambda_i = \omega_i = 1/3$. For the remaining values we bound $\Psi(\bar{\omega}\,, \bar{\lambda}\,, \bar{a}\,, \bar{c}\,)$ with suitable parameters $\bar{a}, \bar{c}$.

Corollary 14. *With $U = \mathcal{U}_\varepsilon(1/3, 1/3), \varepsilon$ small enough*

$$\sum_{\bar{\omega} \notin U, \bar{\lambda} \in U, \lambda_i > 0} N(\bar{w}, \bar{l})/N_0 < C \cdot 3^{(1-\gamma)n}.$$

Proof. Let $\varepsilon > 0$ be such that $\Psi(\bar{\omega}, \bar{\lambda}, \bar{a}, \bar{c}) \leq 3^{1-\gamma}$ for $\bar{a}, \bar{c}$ as specified in Lemma 13 and $\bar{\omega}, \bar{\lambda} \in U$. Let $\bar{\omega} \notin U, \bar{\lambda} \in U$. Using Theorem 9 (2) and some elementary consideration we get that $\Psi(\bar{\omega}, \bar{\lambda}, \bar{a}, \bar{c}) \leq 3^{1-\gamma} - \delta$ for some $\bar{a}, \bar{c}$. This implies the claim as in the proof of Corollary 10.

Theorem 15 is proved by Laplace method following [5], page 71 ff applied to $\exp[\ln \Psi(\bar{\omega}, \bar{\lambda})n]$. We use Lemma 13 and Corollary 12. The factor $1/n^2$ from Corollary 12 is necessary because in the discrete case we need to consider Riemannian sums approximating the integral. The Laplace method leads to an additional explicit factor. However, the constant C is difficult to determine here as N_0 is given up to a constant factor, for example.

Theorem 15. *Let $U = \mathcal{U}_\varepsilon(1/3, 1/3)$. There is an $\varepsilon > 0$ such that* $\sum_{\bar{\lambda}, \bar{\omega} \in U} N(\bar{w}, \bar{l})/N_0 < C \cdot 3^{(1-\gamma)n}$.

Proof of Theorem 5. Combine Theorem 15, Corollary 10, Corollary 14. Terms with an $l_i = 0$ do not add substantially to the sum.

4 Structure of the Proof of Theorem 9

We use the notation $\bar{x} = (x_0, x_1, x_2), \bar{y} = (y_0, y_1, y_2)$ and define $\mathrm{OPT}(\bar{x}, \bar{y}, s) = \mathrm{OPT}_1(\bar{x}, s) \cdot \mathrm{OPT}_2(\bar{x}, \bar{y}, s) \cdot \mathrm{OPT}_3(\bar{y}, s)$ with

$$\mathrm{OPT}_1(\bar{x}, s) = \sum_{i=0}^{2} \frac{q(sx_i)}{q(s)}, \qquad \mathrm{OPT}_2(\bar{x}, \bar{y}, s) = \left(\frac{1}{\sum_{i=0}^{2} x_i y_i} \right)^Q, \sum_{i=0}^{2} x_i y_i > 0$$

$$\mathrm{OPT}_3(\bar{y}, s) = (y_0 + y_1 + y_2)^Q + 2 \cdot \left(y_0^2 + y_1^2 + y_2^2 - y_0 y_1 - y_0 y_2 - y_1 y_2\right)^{1/2 \cdot Q}$$

Observe that $\mathrm{OPT}(1, 1, 1, 1, 1, 1, s) = 3(1/3)^Q 3^Q = 3 = \mathrm{OPT}_1(1, 1, 1, s)$, $\mathrm{OPT}(1, 0, 0, 1, 0, 0, s) = 1 \cdot (1/1)^Q \cdot 3 = 3 = \mathrm{OPT}_3(1, 0, 0, s)$. The following lemma shows the idea of OPT. It follows by considering the 3 factors in the definition of Ψ one by one, observing that x^γ is concave as $\gamma < 1$.

Lemma 16 *Given $\bar{\lambda} > 0$ and let λ be the maximum of the λ_i. Let $a_i, c_i > 0$ be such that $P_i := a_i c_i = \lambda_i / \lambda$. Then $\Psi := \Psi(\bar{\omega}, \bar{\lambda}, \bar{a} \cdot s, \bar{c}) \leq \frac{1}{3^\gamma} OPT(\bar{a}, \bar{c}, s)$.*

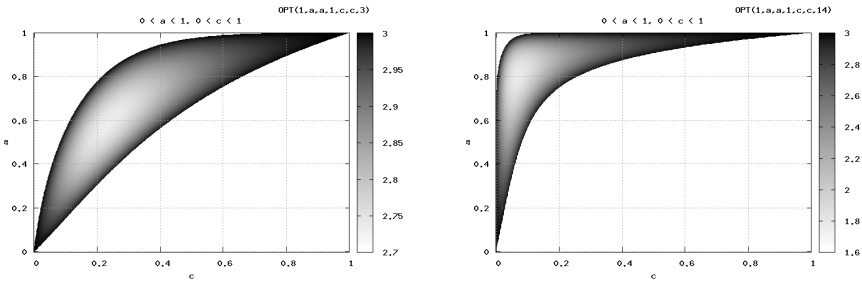


Fig. 1. $\mathrm{OPT}(1, a, a, 1, c, c, s)$ for $0 \leq a \leq 1, 0 \leq c \leq 1$ for $s = 3$ and $s = 14$

The following picture shows $\mathrm{OPT}(1, a, a, 1, c, c, s)$, $0 \leq a, c \leq 1$. The $\leq 3-$area is dark. It gets smaller for increasing k (or s.) We have a path from $a = c = 0$ to $a = c = 1$ through this area. Therefore, for all P with $0 \leq P \leq 1$ we have $0 \leq a, c \leq 1$ with $P = ac$ such that $\mathrm{OPT}(1, a, a, 1, c, c, s) \leq 3$. In the notation of Lemma 16 this corresponds to $\lambda_0 \geq \lambda_1 = \lambda_2$ (and visualizes Theorem 9 for this case.) The following four lemmas show how to walk from $a = c = 0$ to $a = 1, c = 1$ through the $< 3-$area based on simple functions such that proofs are possible. Then Theorem 9 follows based on Lemma 16. The proofs are by elementary though involved calculus.

Lemma 17 *Let* $s \geq 8, A(x) = A(x, s) := (7/10)Q \cdot x$.
(a) $OPT(y) := OPT\,(1, A(y), A(y), 1, y, y, s)$ *is strictly decreasing for* $0 \leq y \leq 1/(2Q)$, $OPT(0) = 3$. *(b) Given* $0 \leq y \leq 1/(2Q)$, $OPT(z) := OPT(1, A(y + z), A(y - z), 1, y + z, y - z, s)$ *is decreasing in* $0 \leq z \leq y$.

Lemma 18 *Let* $s \geq 7$ *, and* $\frac{7}{20} \leq A \leq 1 - \frac{1}{Q}$. *Then* $OPT(z) :=$
$OPT(1, A, A, 1, 1/(2Q) + z, 1/(2Q) - z\,, s) \leq 3 - \delta$ *for* $0 \leq z \leq 1/(2Q)$.

Lemma 19 *Let* $s \geq 7$, *and* $1/(2Q) \leq C \leq 1/2$. *Then* $OPT(z) :=$
$OPT(1, 1 - 1/Q, 1 - 1/Q, 1, C + z, C - z\,, s) \leq 3 - \delta$ *for* $0 \leq z \leq C$.

Lemma 20 *Let* $s \geq 15$ *and* $A(x) = A(x, s) := 1 + 7/(10Q) \cdot x - 7/(10Q)$. *(a)* $OPT(y) := OPT(1, A(y), A(y), 1, y, y, s)$ *is strictly increasing in* $4/10 \leq y \leq 1$, $OPT(1) = 3$. *(b) Given* $4/10 \leq y \leq 1$, $OPT(z) :=$
$OPT(1, A(y + z), A(y - z), 1, y + z\,, y - z\,, s)$ *decreases in* $0 \leq z \leq \min\{y, 1-y\}$.

5 Uniquely Extendible Constraints

A uniquely extendible constraint C on a given domain D is a function from D^k to true, false with the following restriction: For any argument list with a gap at an arbitrary position, like $(d_1, \ldots d_{i-1}, -, d_{i+1}, \ldots, d_k)$ there is a unique $d \in D$ such that $C(d_1, \ldots d, \ldots, d_k)$ evaluates to true. Note that $C(d_1, \ldots, d, \ldots, d_k) =$ true implies that $C(d_1, \ldots, d', \ldots, d_k) =$ false for $d \neq d'$. The random constraint is a

uniform random member from the set of all uniquely extendible constraints over D. Let Γ be the set of all such constraints. Typical examples of such constraints are linear equations with k variables, modulo $|D|$. A threshold result in the sense of Friegut–Bourgain can be proved based on symmetry properties of uniquely extendible constraints. Given a set of n variables a clause is an ordered k-tuple of variables equipped with a uniquely extendible constraint.

Given a k-tuple a of values from D and another k-tuple b differing from a in exactly i, $0 \leq i \leq k$, slots, we let p_i be the probability that the random constraint is true under b conditional on the event that it is true under a. It is known that p_i is well defined. Let $p(z) = 1/d \cdot [(1+z)^k + (d-1)(1-z/(d-1))^k]$. The next lemma shows that $p(z)$ serves the same purpose as $r(\bar{x})$ before.

Lemma 21 *(a) (From [8])* $p_0 = 1,\ p_{i+1} = \frac{1}{d-1}(1-p_i)$ *(b)* $p(z) = \sum_i \binom{k}{i} p_i \cdot z^i$.

Proof. (b) We need to show that $p_i = \frac{1}{d}\left(1 + (-1)^i \left(\frac{1}{d-1}\right)^{i-1}\right)$. This holds for $i = 0, i = 1$. For $i > 1$ we get by induction:

$$p_i = \frac{1}{d-1}(1-p_{i-1}) = \frac{1}{d-1}\left(1 - \frac{1}{d}\left(1 + (-1)^{i-1}\left(\frac{1}{d-1}\right)^{i-2}\right)\right) =$$

$$= \frac{1}{d-1} - \frac{1}{d(d-1)} - \frac{1}{d}(-1)^{i-1}\left(\frac{1}{d-1}\right)^{i-1} = \frac{1}{d}\left(1 + (-1)^i\left(\frac{1}{d-1}\right)^{i-1}\right).$$

With this lemma we can define a function analogous to Ψ and OPT and proceed quite similarly to the mod 3−case. Additional issues arise because the structure of $p(z)$ is less symmetric than the structure of $r(\bar{x})$. The details must be omitted here.

References

1. Achlioptas, D., Moore, C.: Random k-SAT: Two Moments Suffice to Cross a Sharp Threshold. SIAM J. Comput. 36(3), 740–762 (2006)
2. Achlioptas, D., Ibrahimi, M., Kanoria, Y., Kraning, M., Molloy, M., Montanari, A.: The Set of Solutions of Random XORSAT Formulae. In: Proceedings SoDA 2012 (2012)
3. Bhattacharya, N., Ranga Rao, R.: Normal Approximation and Asymptotic Expansions. Robert E. Krieger Publishing Company (1986)
4. Braunstein, A., Mezard, M., Zecchina, R.: Survey propagation: an algorithm for satisfiability. arXiv:cs/0212002
5. de Bruijn, N.G.: Asymptotic Methods in Analysis. North Holland (1958)
6. Coja-Oghlan, A., Pachon-Pinzon, A.Y.: The Decimation Process in Random k-SAT. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 305–316. Springer, Heidelberg (2011)
7. Connamacher, H.: Exact thresholds for DPLL on random XOR-SAT and NP-complete extensions of XOR-SAT. Theoretical Computer Science (2011)

8. Connamacher, H., Molloy, M.: The exact satisfiability threshold for a potentially in tractable random constraint satisfaction problem. In: Proceedings 45th FoCS 2004, pp. 590–599 (2004)
9. Creignou, N., Daudé, H.: The SAT-UNSAT transition for random constraint satisfaction problems. Discrete Mathematics 309(8), 2085–2099
10. Diaz, J., et al.: On the satisfiability threshold of formulas with three literals per clause. Theoretical Computer Science 410, 2920–2934 (2009)
11. Dietzfelbinger, M., Goerdt, A., Mitzenmacher, M., Montanari, A., Pagh, R., Rink, M.: Tight Thresholds for Cuckoo Hashing via XORSAT. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part I. LNCS, vol. 6198, pp. 213–225. Springer, Heidelberg (2010); arXiv:cs/0912.0287 (2009)
12. Dubois, O., Mandler, J.: The 3−XORSAT satisfiability threshold. In: Proceedings 43rd FoCS, p. 769 (2003)
13. Durrett, R.: Probability Theory: Theory and Examples. Wadsworth and Brooks (1991)
14. Friedgut, E.: Hunting for sharp thresholds. Random Struct. Algorithms 26(1-2), 37–51 (2005)
15. Goerdt, A.: On Random Betweenness Constraints. Combinatorics, Probability and Computing 19(5-6), 775–790 (2010)
16. Goerdt, A., Falke, L.: Satisfiability thresholds beyond k−XORSAT. arXiv:cs/1112.2118
17. Hastad, J.: Some optimal inapproximability results. J. ACM 48, 798–859 (2001)
18. Kolchin, V.F.: Random graphs and systems of linear equations in finite fields. Random Structures and Algorithms 5, 425–436 (1995)
19. Luby, M., Mitzenmacher, M., Shokrollahi, A., Spielman, D.A.: Efficient erasure coeds. IEEE Trans. Inform. Theory 47(2), 569–584 (2001)
20. Meisels, A., Shimony, S.E., Solotorevsky, G.: Bayes Networks for estimating the number of solutions to a CSP. In: Proceedings AAAI 1997, pp. 179–184 (1997)
21. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press (2005)
22. Molloy, M.: Cores in random hypergraphs and boolean formulas. Random Stuctures and Algorithms 27, 124–135 (2005)
23. Molloy, M.: Models for Random Constraint Satisfaction Problems. SIAM J. Comput. 32(4), 935–949 (2003)
24. Puyhaubert, V.: Generating functions and the satisfiability threshold. Discrete Mathematics and Theoretical Computer Science 6, 425–436 (2004)
25. Richardson, T.J., Urbanke, R.: Modern Coding Theory. Cambridge University Press (2008)

Finding Vertex-Surjective Graph Homomorphisms*

Petr A. Golovach[1], Bernard Lidický[2,3], Barnaby Martin[1], and Daniël Paulusma[1]

[1] School of Engineering and Computing Sciences, Durham University Science Laboratories, South Road, Durham DH1 3LE, UK
{petr.golovach,barnaby.martin,daniel.paulusma}@durham.ac.uk
[2] Charles University, Faculty of Mathematics and Physics, Malostranské nám. 2/25, 118 00, Prague, Czech Republic
bernard@kam.mff.cuni.cz
[3] Department of Mathematics, University of Illinois Urbana, IL 61801, USA

Abstract. The Surjective Homomorphism problem is to test whether a given graph G called the guest graph allows a vertex-surjective homomorphism to some other given graph H called the host graph. The bijective and injective homomorphism problems can be formulated in terms of spanning subgraphs and subgraphs, and as such their computational complexity has been extensively studied. What about the surjective variant? Because this problem is NP-complete in general, we restrict the guest and the host graph to belong to graph classes $\mathcal{G}$ and $\mathcal{H}$, respectively. We determine to what extent a certain choice of $\mathcal{G}$ and $\mathcal{H}$ influences its computational complexity. We observe that the problem is polynomial-time solvable if $\mathcal{H}$ is the class of paths, whereas it is NP-complete if $\mathcal{G}$ is the class of paths. Moreover, we show that the problem is even NP-complete on many other elementary graph classes, namely linear forests, unions of complete graphs, cographs, proper interval graphs, split graphs and trees of pathwidth at most 2. In contrast, we prove that the problem is fixed-parameter tractable in k if $\mathcal{G}$ is the class of trees and $\mathcal{H}$ is the class of trees with at most k leaves, or if $\mathcal{G}$ and $\mathcal{H}$ are equal to the class of graphs with vertex cover number at most k.

1 Introduction

We consider undirected finite graphs that are *simple*, i.e., have no loops and no multiple edges. A graph is denoted $G = (V_G, E_G)$, where V_G is the set of vertices and E_G is the set of edges. A *homomorphism* from a graph G to a graph H is a mapping $f : V_G \to V_H$ that maps adjacent vertices of G to adjacent vertices of H, i.e., $f(u)f(v) \in E_H$ whenever $uv \in E_G$. Graph homomorphisms are widely

* The work was supported by EPSRC (EP/G043434/1 and EP/G020604/1) and the Royal Society (JP090172). The 2nd author was supported by Charles University as GAUK 95710.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 160–171, 2012.

studied within the areas of graph theory and algorithms; for a survey we refer to the monograph of Hell and Nešetřil [18]. The HOMOMORPHISM problem is to test whether there exists a homomorphism from a graph G called the *guest graph* to a graph H called the *host graph*. If H is restricted to be in the class of complete graphs (graphs with all possible edges), then this problem is equivalent to the COLORING problem. The latter problem is to test whether a graph G allows a *k-coloring* for some given k, i.e., a mapping $c : V_G \to \{1, \ldots, k\}$, such that $c(u) \neq c(v)$ whenever $uv \in E_G$. This is a classical NP-complete problem [15]. Hence, the HOMOMORPHISM problem is NP-complete in general, and it is natural to restrict the input graphs to belong to some special graph classes.

We let $\mathcal{G}$ denote the class of guest graphs and $\mathcal{H}$ the class of host graphs that are under consideration, and denote the corresponding decision problem by $(\mathcal{G}, \mathcal{H})$-HOMOMORPHISM. If G or H is the class of all graphs, then we use the notation "$-$" to indicate this. If $\mathcal{G} = \{G\}$ or $\mathcal{H} = \{H\}$, we write G and H instead of $\mathcal{G}$ and $\mathcal{H}$, respectively, The Hell-Nešetřil dichotomy theorem [17] states that $(-, H)$-HOMOMORPHISM is solvable in polynomial time if H is bipartite, and NP-complete otherwise. In the context of graph homomorphisms, a graph F is called a *core* if there exists no homomorphism from F to any proper subgraph of F. Dalmau et al. [5] proved that the $(\mathcal{G}, -)$-HOMOMORPHISM problem can be solved in polynomial time if all cores of the graphs in $\mathcal{G}$ have bounded treewidth. Moreover, Grohe [16] showed that under the assumption FPT $\neq$ W[1], the problem can be solved in polynomial time if and only if this condition holds.

As a homomorphism f from a graph G to a graph H is a (vertex) mapping, we may add further restrictions, such as requiring it to be *bijective*, *injective*, or *surjective* i.e., for each $x \in V_H$ there exists exactly one, at most one, or at least one vertex $u \in V_G$ with $f(u) = x$, respectively. The decision problems corresponding to the first and second variant are known as the SPANNING SUBGRAPH ISOMORPHISM and SUBGRAPH ISOMORPHISM problem, respectively. As such, these two variants have been well studied in the literature. For example, the bijective variant contains the problem that is to test whether a graph contains a Hamiltonian cycle as a special case. In our paper, we research the third variant, which leads to the following decision problem:

SURJECTIVE HOMOMORPHISM
Instance: two graphs G and H.
Question: does there exist a surjective homomorphism from G to H?

If the guest G is restricted to a graph class $\mathcal{G}$ and the host H to a graph class $\mathcal{H}$, then we denote this problem by SURJECTIVE $(\mathcal{G}, \mathcal{H})$-HOMOMORPHISM. Fixing the host side to a single graph H yields the SURJECTIVE $(-, H)$-HOMOMORPHISM problem. This problem is NP-complete already when H is nonbipartite. This follows from a simple reduction from the corresponding $(-, H)$-HOMOMORPHISM problem, which is NP-complete due to the Hell-Nešetřil dichotomy theorem [17]; we replace an instance graph G of the latter problem by the disjoint union $G+H$ of G and H, and observe that G allows an homomorphism to H if and only if $G+H$ allows a surjective homomorphism to H. For bipartite host graphs H, the complexity classification of SURJECTIVE $(-, H)$-HOMOMORPHISM is still open,

although some partial results are known. For instance, the problem can be solved in polynomial time whenever H is a tree. This follows from a more general classification that also includes trees in which the vertices may have self-loops [14]. On the other hand, there exist cases of bipartite host graphs H for which the problem is NP-complete, e.g., when H is the graph obtained from a 6-vertex cycle with one distinct path of length 3 added to each of its six vertices [2]. Recently, the Surjective $(-,H)$-Homomorphism problem has been shown to be NP-complete when H is a 4-vertex cycle with a self-loop in every vertex [20]. Note that in our paper we only consider simple graphs. For a survey on the Surjective $(-,H)$-Homomorphism problem from a constraint satisfaction point of view we refer to the paper of Bodirsky, Kára and Martin [2]. Below we discuss some other concepts that are closely related to surjective homomorphisms.

A homomorphism f from a graph G to a graph H is *locally surjective* if f becomes surjective when restricted to the neighborhood of every vertex u of G, i.e., $f(N_G(u)) = N_H(f(u))$. The corresponding decision is called the Role Assignment problem which has been classified for any fixed host H [11]. Any locally surjective homomorphism is surjective if the host graph is connected but the reverse implication is not true in general. For more on locally surjective homomorphisms and the locally injective and bijective variants, we refer to the survey of Fiala and Kratochvíl [9].

Let H be an induced subgraph of a graph G. Then a homomorphism f from a graph G to H is a *retraction* from G to H if $f(h) = h$ for all $h \in V_H$. In that case we say that G *retracts to* H. By definition, a retraction from G to H is a surjective homomorphism from G to H. Retractions are well studied; see e.g. the recent complexity classification of Feder et al. [7] for the corresponding decision problem when H is a fixed pseudoforest. In particular, polynomial-time algorithms for retractions have been proven to be a useful subroutine for obtaining polynomial-time algorithms for the Surjective $(-,H)$-Homomorphism problem [14].

We emphasize that a surjective homomorphism is *vertex-surjective* as opposed to the stronger condition of being edge-surjective. A homomorphism from a graph G to a graph H is called *edge-surjective* or a *compaction* if for any edge $xy \in E_H$ there exists an edge $uv \in E_G$ with $f(u) = x$ and $f(v) = y$. If f is a compaction from G to H, we also say that G *compacts* to H. The Compaction problem is to test whether a graph G compacts to a graph H. Vikas [21,22,23] determined the computational complexity of $(-,H)$-Compaction for several classes of fixed host graphs H. Very recently, Vikas [24] considered $(-,H)$-Compaction for guest graphs belonging to some restricted graph class.

Our Results. We study the Surjective $(\mathcal{G},\mathcal{H})$-Homomorphism problem for several graph classes $\mathcal{G}$ and $\mathcal{H}$. We observe that while this problem is polynomial-time solvable when a host graph is a path, it becomes NP-complete if we restrict the guests to be paths instead of the hosts. We also show that the problem is NP-complete when both $\mathcal{G}$ and $\mathcal{H}$ are restricted to trees of pathwidth at most 2, and when both $\mathcal{G}$ and $\mathcal{H}$ are linear forests. These results are in contrast to the polynomial-time result of Grohe [16] on $(\mathcal{G},-)$-Homomorphism for graph classes $\mathcal{G}$ that consists of graphs, the cores of which have bounded treewidth.

They are also in contrast to the polynomial-time result [14] on Surjective $(-, H)$-Homomorphism when H is any fixed tree.

Due to the hardness for graphs of bounded treewidth, it is natural to consider other width parameters such as the clique-width of a graph. For this purpose we first consider the class of complete graphs that are exactly those graphs that have clique-width 1. We observe that the Surjective $(\mathcal{G}, \mathcal{H})$-Homomorphism can be solved in polynomial time when $\mathcal{G}$ is the class of complete graphs, whereas the problem becomes NP-complete when we let $\mathcal{G}$ and $\mathcal{H}$ consist of the unions of complete graphs. We then focus on graphs that have clique-width at most two. This graph class is equal to the class of cographs [4]. There exist only a few natural problems that are difficult on cographs. We prove that Surjective $(\mathcal{G}, \mathcal{H})$-Homomorphism, where $\mathcal{G}$ and $\mathcal{H}$ are equal to the class of connected cographs, is one of these. We also consider proper interval graphs. This graph class has unbounded tree-width and contains the classes of complete graphs and paths. Because they are "path-like", often problems that are difficult for general graphs are tractable for proper interval graphs. In an attempt to generalize our polynomial-time result for Surjective $(\mathcal{G}, \mathcal{H})$-Homomorphism when $\mathcal{G}$ is the class of complete graphs, or when $\mathcal{H}$ is the class of paths, we consider connected proper interval graphs. It turns out that Surjective $(\mathcal{G}, \mathcal{H})$-Homomorphism is NP-complete even when $\mathcal{G}$ and $\mathcal{H}$ consist of these graphs. Our last hardness result shows that the problem is also NP-complete when $\mathcal{G}$ and $\mathcal{H}$ are equal to the class of split graphs. All hardness results can be found in Section 3.

Table 1. Complexity of $(\mathcal{G}, \mathcal{H})$-Surjective Homomorphism

$\mathcal{G}$	$\mathcal{H}$	Complexity	
complete graphs	all graphs	polynomial	Proposition 1 (i)
all graphs	paths	polynomial	Proposition 1 (ii)
paths	all graphs	NP-complete	Theorem 1 (i)
linear forests	linear forests	NP-complete	Theorem 1 (ii)
unions of complete graphs	unions of complete graphs	NP-complete	Theorem 1 (iii)
connected cographs	connected cographs	NP-complete	Theorem 1 (iv)
trees of $\mathbf{pw} \leq 2$	trees of $\mathbf{pw} \leq 2$	NP-complete	Theorem 1 (v)
split graphs	split graphs	NP-complete	Theorem 1 (vi)
connected proper interval graphs	connected proper interval graphs	NP-complete	Theorem 1 (vii)
trees	trees with k leaves	FPT in k	Theorem 2
graphs of $\mathbf{vc} \leq k$	graphs of $\mathbf{vc} \leq k$	FPT in k	Theorem 3

To complement our hardness results, we show in Section 4 that Surjective $(\mathcal{G}, \mathcal{H})$-Homomorphism is fixed-parameter tractable in k, when $\mathcal{G}$ is the class of trees and $\mathcal{H}$ is the class of trees with at most k leaves, and also when $\mathcal{G}$ and $\mathcal{H}$ consist of graphs with vertex cover number at most k. The latter result adds further evidence that decision problems difficult for graphs of bounded treewidth may well be tractable if the vertex cover number is bounded; also see e.g. [1,6,8,10]. Moreover, the vertices of such graphs can be partitioned into two

sets, one of them has size bounded by the vertex cover number and the other one is an independent set. As such, they resemble split graphs with bounded clique number. We refer to Table 1 for a summary of our results. In this table, **pw** and **vc** denote the pathwidth and the vertex cover number of a graph, respectively. In Section 2 we explain these notions and the complexity class FPT. There, we also give the definitions of all the aforementioned graph classes.

2 Definitions and Preliminaries

Let G be a graph. The *open neighborhood* of a vertex $u \in V_G$ is defined as $N_G(u) = \{v \mid uv \in E_G\}$, and its *closed neighborhood* is defined as $N_G[u] = N_G(u) \cup \{u\}$. The degree of a vertex $u \in V_G$ is denoted $d_G(u) = |N_G(u)|$. The *distance* $\mathrm{dist}_G(u, v)$ between a pair of vertices u and v of G is the number of edges of a shortest path between them. The *distance* between a vertex u and a set of vertices $S \subseteq V_G$ is $\mathrm{dist}_G(u, S) = \min\{\mathrm{dist}_G(u, v) | v \in S\}$. We may omit subscripts if this does not create any confusion. The *diameter* of G is defined as $\mathrm{diam}(G) = \max\{\mathrm{dist}_G(u, v) | u, v \in V_G\}$. Let $S \subseteq V_G$. Then the graph $G - S$ is the graph obtained from G by removing all vertices in S. If $S = \{u\}$, we also write $G - u$. The subgraph of G that is *induced* by S has vertex set S and edges uv if and only if $uv \in E_G$. We denote this subgraph by $G[S]$.

A graph is an *interval graph* if intervals of the real line can be associated with its vertices in such a way that two vertices are adjacent if and only if their corresponding intervals overlap. An interval graph is *proper* if it has an interval representation, in which no interval is properly contained in any other interval. The disjoint union of two graphs G and H is denoted $G + H$, and the disjoint union of r copies of G is denoted rG. A *linear forest* is the disjoint union of a collection of paths. We denote the path on n vertices by P_n. A graph is a *cograph* if it does not contain P_4 as an induced subgraph. A *clique* is the vertex set of a complete graph. A vertex set is *independent* if its vertices are mutually non-adjacent. A graph is a *split graph* if its vertex set can be partitioned into a clique and an independent set.

A *tree decomposition* of a graph G is a pair $(\mathcal{X}, T)$ where T is a tree and $\mathcal{X} = \{X_i \mid i \in V_T\}$ is a collection of subsets (called *bags*) of V_G such that the following three conditions are satisfied: (i) $\bigcup_{i \in V_T} X_i = V_G$; (ii) for each edge $xy \in E_G$, the vertices x, y are in a bag X_i for some $i \in V_T$; and (iii) for each $x \in V_G$, the set $\{i \mid x \in X_i\}$ induces a connected subtree of T. The *width* of tree decomposition $(\mathcal{X}, T)$ is $\max_{i \in V_T} \{|X_i| - 1\}$. The *treewidth* of a graph G, denoted $\mathbf{tw}(G)$, is the minimum width over all tree decompositions of G. If in these two definitions we restrict the tree T to be a path, then we obtain the notions of *path decomposition* and *pathwidth* of G denoted $\mathbf{pw}(G)$.

For a graph G, a set $S \subseteq V_G$ is a *vertex cover* of G, if every edge of G has at least one of its two end-vertices in S. Let $\mathbf{vc}(G)$ denote the *vertex cover number*, i.e., the minimum size of a vertex cover of G.

We use the following well-known notion in parameterized complexity, where one considers the problem input as a pair (I, k), where I is the main part and k

the parameter; also see the text book of Flum and Grohe [12]. A problem is *fixed parameter tractable* if an instance (I, k) can be solved in time $O(f(k)n^c)$, where f denotes a computable function, n denotes the size of I, and c is a constant independent of k. The class FPT is the class of all fixed-parameter tractable decision problems.

We finish this section with giving the polynomial-time results from Table 1.

Proposition 1. *The* Surjective $(\mathcal{G},\mathcal{H})$-Homomorphism *problem can be solved in polynomial time in the following two cases: (i) $\mathcal{G}$ is the class of complete graphs and $\mathcal{H}$ is the class of all graphs; (ii) $\mathcal{G}$ is the class of all graphs and $\mathcal{H}$ is the class of paths.*

3 Hard Cases

In contrast to case (ii) of Proposition 1, where the host graphs are assumed to be paths, our problem becomes difficult when the guest graphs are restricted to paths. Our next theorem shows this and the other hardness results of Table 1.

Theorem 1. *The* Surjective $(\mathcal{G},\mathcal{H})$-Homomorphism *problem is* NP*-complete in the following six cases:*

(i) $\mathcal{G}$ is the class of paths and $\mathcal{H}$ is the class of all graphs;
(ii) $\mathcal{G} = \mathcal{H}$ is the class of linear forests;
(iii) $\mathcal{G} = \mathcal{H}$ is the class of disjoint unions of complete graphs;
(iv) $\mathcal{G} = \mathcal{H}$ is the class of connected cographs;
(v) $\mathcal{G} = \mathcal{H}$ is the class of trees of pathwidth at most two;
(vi) $\mathcal{G} = \mathcal{H}$ is the class of split graphs;
(vii) $\mathcal{G} = \mathcal{H}$ is the class of connected proper interval graphs.

Proof. We first prove (i). We reduce from the well-known problem Hamiltonian Path, which is NP-complete [15]. An n-vertex graph H has a Hamiltonian path if and only if there exists a surjective homomorphism from P_n to H. This proves (i).

For showing (ii)-(vii) we need some extra terminology. We say that a multiset $A = \{a_1, \ldots, a_n\}$ of integers is (m, B)*-positive* if $n = 3m$, $\sum_{i=1}^{n} a_i = mB$ and $a_i > 0$ for $i = 1, \ldots, n$. A 3*-partition* of a multiset $A = \{a_1, \ldots, a_n\}$ that is (m, B)-positive for some integers m, B is a partition $S_1, S_2, \ldots, S_m$ of A such that for $1 \leq j \leq m$, $|S_j| = 3$ and $\sum_{a_i \in S_j} a_i = B$. This leads to the problem:

3-Partition
Instance: an (m, B)-positive multiset $A = \{a_1, \ldots, a_n\}$ for some integers m, B;
Question: does A have a 3-partition?

The 3-Partition problem is known to be NP-complete [15] in the strong sense, i.e., it remains hard even if all integers in the input are encoded in unary. This enables us to reduce from this problem in order to show NP-completeness in the cases (ii)-(vii). In each of these six cases we assume that $A = \{a_1, \ldots, a_n\}$ is a (m, B)-positive multiset for some integers m, B. We now prove (ii)-(vii).

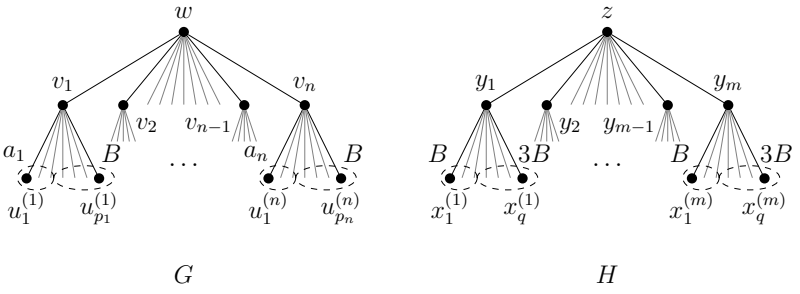


Fig. 1. The trees G and H constructed in the proof of (v).

(ii) For $i = 1, \ldots, n$, let $p_i = a_i + B$, and let $q = 4B$. Let G be the linear forest $G_1 + \cdots + G_n$, where G_i is isomorphic to P_{p_i} for $i = 1, \ldots, n$. Let H be the linear forest $H_1 + \cdots + H_m = mP_q$. We claim that A has a 3-partition if and only if there exists a surjective homomorphism from G to H.

Suppose that $S_1, \ldots, S_m$ is a 3-partition of A. For each $1 \leq j \leq m$, we consider the connected components $G_{i_1}, G_{i_2}, G_{i_3}$ of G such that $S_j = \{a_{i_1}, a_{i_2}, a_{i_3}\}$. We map the vertices of G_{i_1} to the first p_{i_1} vertices of H_j according to the path order, and similarly the vertices of G_{i_2} to the next p_{i_2} vertices of H_j, and the vertices of G_{i_3} to the last p_{i_3} vertices of H_j. Because $p_{i_1} + p_{i_2} + p_{i_3} = a_{i_1} + a_{i_2} + a_{i_3} + 3B = 4B = q$ for $i = 1, \ldots, n$, we obtain a surjective homomorphism from G to H in this way.

Now suppose that f is a surjective homomorphism from G to H. We observe that $|V_G| = |V_H| = 4mB$. Hence, f is also injective. Because f is a homomorphism, f must map all vertices of each connected component of G to the same connected component of H. Let $1 \leq j \leq m$, and let $G_{i_1}, \ldots, G_{i_s}$ be the connected components of G that are mapped to H_j. Because $|V_{H_j}| = 4B$ and every connected component of G contains at least $B + 1$ vertices, we find that $s \leq 3$. Because G has $3m$ connected components, we then find that $s = 3$. Because f is injective, $a_{i_1} + a_{i_2} + a_{i_3} + 3B = p_{i_1} + p_{i_2} + p_{i_3} = q = 4B$. Hence, $a_{i_1} + a_{i_2} + a_{i_3} = B$ and we let $S_j = \{a_{i_1}, a_{i_2}, a_{i_3}\}$. This means that we obtain the partition $S_1, \ldots, S_m$ of A that is a 3-partition.

(iii) and **(iv)** We use the constructions from the proof of (ii). For (iii), we replace each path in G and H by a clique of the same size. For (iv), we replace each path by a clique of the same size. We also add a vertex v in G adjacent to all other vertices of G, and a vertex x in H adjacent to all other vertices of H. The resulting graphs are connected cographs. We observe that every homomorphism maps v to x. To finish the proofs we use the same arguments as the ones used to prove (ii).

(v) For $i = 1, \ldots, n$, let $p_i = a_i + B$, and let $q = 4B$. We construct two trees G and H. We first construct G:

- for $i = 1, \ldots, n$, introduce p_i vertices $u_1^{(i)}, \ldots, u_{p_i}^{(i)}$ and a vertex v_i adjacent to $u_1^{(i)}, \ldots, u_{p_i}^{(i)}$;
- add a new vertex w and make it adjacent to $v_1, \ldots, v_n$.

We now construct H:

- for $j = 1, \ldots, m$, introduce q vertices $x_1^{(j)}, \ldots, x_q^{(j)}$ and a vertex y_j adjacent to $x_1^{(j)}, \ldots, x_q^{(j)}$;
- add a new vertex z and make it adjacent to $y_1, \ldots, y_m$.

The trees G and H are displayed in Figure 1. For G we take the path decomposition with bags $\{u_h^{(i)}, v_i, w\}$ to find that $\mathbf{pw}(G) \leq 2$. Similarly, we find that $\mathbf{pw}(H) \leq 2$. We claim that A has a 3-partition if and only if there is a surjective homomorphism from G to H.

First suppose that $S_1, \ldots, S_m$ is a 3-partition of A. We define f as follows. We set $f(w) = z$. Then for $j = 1, \ldots, m$, we consider the set $S_j = \{a_{i_1}, a_{i_2}, a_{i_3}\}$. We let f map the vertices $v_{i_1}, v_{i_2}, v_{i_3}$ to y_j. Then we let f map the vertices $u_1^{(i_1)}, \ldots, u_{p_{i_1}}^{(i_1)}$ consecutively to the first p_{i_1} vertices of the set $\{x_1^{(j)}, \ldots, x_q^{(j)}\}$, the vertices $u_1^{(i_2)}, \ldots, u_{p_{i_2}}^{(i_2)}$ to the next p_{i_2} vertices of this set, and finally, the vertices $u_1^{(i_3)}, \ldots, u_{p_{i_3}}^{(i_3)}$ to the last p_{i_3} vertices of the set. Because $p_{i_1} + p_{i_2} + p_{i_3} = a_{i_1} + a_{i_2} + a_{i_3} + 3B = 4B = q$, we find that f is a surjective homomorphism from G to H.

Now suppose that f is a surjective homomorphism from G to H. We observe that $f(w) = z$, because all vertices of G must be mapped at distance at most two from $f(w)$. Consequently, f maps every v-vertex to a y-vertex, and every u-vertex to an x-vertex. The number of u-vertices is $p_1 + \ldots + p_n = a_1 + \ldots + a_n + nB = 4mB$, which is equal to the number of x-vertices. Hence f maps the u-vertices bijectively to the x-vertices. Moreover, if $f(v_i) = y_j$, then f maps the vertices $u_1^{(i)}, \ldots, u_{p_i}^{(i)}$ to the vertices from the set $\{x_1^{(j)}, \ldots, x_q^{(j)}\}$. For $j = 1, \ldots, m$, let $v_{i_1}, \ldots, v_{i_s}$ be the vertices mapped to y_j. Because $p_i > B$ for all $1 \leq i \leq n$, we find that $s \leq 3$. Then, because $n = 3m$, we conclude that $s = 3$. Because f maps bijectively $\{u_1^{(i_1)}, \ldots, u_{p_{i_1}}^{(i_1)}\} \cup \{u_1^{(i_2)}, \ldots, u_{p_{i_2}}^{(i_2)}\} \cup \{u_1^{(i_3)}, \ldots, u_{p_{i_3}}^{(i_3)}\}$ to $\{x_1^{(j)}, \ldots, x_q^{(j)}\}$, we find that $a_{i_1} + a_{i_2} + a_{i_3} + 3B = p_{i_1} + p_{i_2} + p_{i_3} = q = 4B$, and consequently, $a_{i_1} + a_{i_2} + a_{i_3} = B$. We set $S_j = \{a_{i_1}, a_{i_2}, a_{i_3}\}$. It remains to observe that $S_1, \ldots, S_m$ is a 3-partition of A. This completes the proof of (v).

The proofs of **(vi)** and **(vii)** are based on similar arguments and have been omitted. □

4 Tractable Cases

By Theorem 1 (v), Surjective Homomorphism is NP-complete when G and H are restricted to be trees. Here, we prove that the problem is FPT for trees when parameterized by the number of leaves in H. We first need some additional terminology. Let T be a tree. Then we may fix some vertex of T and call it the *root* of T. We observe that the root defines a parent-child relation between adjacent vertices. This enables us to define for a vertex u of T the tree G_u, which is the subtree of T that is induced by u and all its descendants in T; we fix u to be the root of G_u. For a child v of u, we let G_{uv} denote the subtree of G induced by u and the set of all descendants of v in T; we fix u to be the root of G_{uv}.

Theorem 2. *Testing if there is a surjective homomorphism from an n-vertex tree G to an m-vertex tree H with k leaves can be done in $O(2^{2k}nm^2)$ time.*

Proof. We use dynamic programming. If H has one vertex the claim is trivial. Assume that H has at least one edge. Let L be the set of the leaves of H. First, we fix a root r of G. For each vertex $u \in V_G$, we construct a table that contains a number of records $R = (x, S)$ where $x \in V_H$ and $S \subseteq L$. A pair (x, S) is a record for u if and only if there exists a homomorphism h from G_u to H such that $h(u) = x$ and $S \subseteq h(V_{G_u})$. We also construct a similar table for each edge $uv \in E_G$. Then a pair (x, S) is a record for uv if and only if there exists a homomorphism h from G_{uv} to H such that $h(u) = x$ and $S \subseteq h(V_{G_{uv}})$. The key observation is that a homomorphism f from G to H is surjective if and only if $L \subseteq f(V_G)$, i.e., if and only if the table for r contains at least one record (z, L).

We construct the tables as follows. We start with the leaves in G not equal to r (should r be a leaf). Their tables are constructed straightforwardly. Suppose that we have not constructed the table for a vertex u, while we have constructed the tables for all children $v_1, \ldots, v_p$ of u. Then we first determine the table for each edge uv_i by letting it consist of all records (x, S) such that

- (y, S) with $y \in N_H(x)$ is in the table for v_i;
- $x \in L$ and $(y, S \setminus \{x\})$ with $y \in N_H(x)$ is in the table for v_i.

To construct the table for u, we consecutively construct auxiliary tables for $i = 1, \ldots, p$. The table for $i = 1$ is the table for uv_1. The table for $i \geq 2$ consists of the records (x, S) such that $S = S' \cup S''$, (x, S') is in the table for $i - 1$ and (x, S'') is in the table for uv_i. The table for u is the table constructed for $i = p$.

The correctness of the algorithm follows from its description. We observe that each table contains at most $m2^k$ records and can be constructed in $O(2^{2k} \cdot m^2)$ time. Because we construct $O(n)$ tables (including the auxiliary ones), our algorithms runs in $O(2^{2k} \cdot nm^2)$ time. This completes the proof of Theorem 2. □

We now prove that Surjective Homomorphism is FPT when parameterized by the vertex cover number of G and H. The following approach has been successful before [8,10]. The idea is to reduce a problem to an integer linear programming problem that is FPT when parameterized by the number of variables. Therefore, we consider the p-Variable Integer Linear Programming Feasibility problem that has as input a $q \times p$ matrix A with integer elements and an integer vector $b \in \mathbb{Z}^q$ and that is to decide whether there exists a vector $x \in \mathbb{Z}^p$ such that $A \cdot x \leq b$. Lenstra [19] showed that this problem is FPT when parameterized by p. The best running time is due to Frank and Tardos [13].

Lemma 1 ([13]). *The p-Variable Integer Linear Programming Feasibility problem can be solved using $O(p^{2.5p+o(p)} \cdot L)$ arithmetic operations and space polynomial in L, where L is the number of bits of the input.*

Theorem 3. *Testing if there is a surjective homomorphism from an n-vertex graph G with $\mathbf{vc}(G) \leq k$ to an m-vertex graph H with $\mathbf{vc}(H) \leq k$ can be done in $2^{2^{O(k)}}(nm)^{O(1)}$ time.*

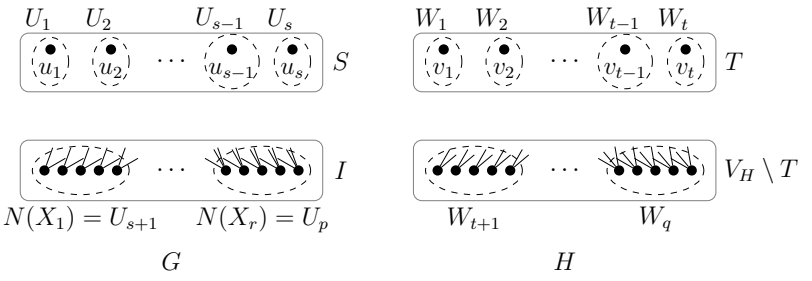


Fig. 2. The graphs G and H as considered in the proof of Theorem 3

Proof. Let G be an n-vertex graph with a vertex cover $S = \{u_1, \ldots, u_s\}$ of size $s \leq k$. Then $I = V_G \setminus S$ is an independent set. For every subset $X \subseteq S$, we define $N(X)$ as the set of vertices in I that all have neighborhood X, i.e., $N(X) = \{u \in I \mid N(u) = X\}$. Note that $N(\emptyset)$ is the set of isolated vertices in I.

Let $X_1, \ldots, X_r \subseteq S$ be the sets with $N(X_i) \neq \emptyset$. We let $p = s + r$ and define sets $U_1, \ldots, U_p$ where $U_i = \{u_i\}$ for $i = 1, \ldots, s$ and $U_i = N(X_{i-s})$ for $i = s+1, \ldots, p$. We observe that $p \leq k + 2^k$ and that $U_1, \ldots, U_p$ is a partition of V_G, where each U_i is an independent set. Moreover, a vertex $v \in U_i$ is adjacent to a vertex $w \in U_j$ if and only if each vertex of U_i is adjacent to each vertex of U_j. In that case, we say that U_i is *adjacent* to U_j. We display G in Figure 2.

Let H be an m-vertex graph with a vertex cover $T = \{v_1, \ldots, v_t\}$ of size $t \leq k$. Then $J = V_H \setminus T$ is an independent set, and for each $Y \subseteq T$ we define $N(Y) = \{z \in J \mid N(z) = Y\}$. Then we define $q \leq k + 2^k$ sets $W_1, \ldots, W_q$ where $W_j = \{v_j\}$ for $j = 1, \ldots, t$ and $W_j = N(Y_{j-t})$ for $j = t+1, \ldots, q$. We also display H in Figure 2. The observations that we made for the U-sets are also valid for the W-sets.

Now we introduce integer variables x_{ij} for $1 \leq i \leq p$ and $1 \leq j \leq q$, and observe that there is a surjective mapping (not necessarily a homomorphism) $f \colon V_G \to V_H$ such that x_{ij} vertices of U_i are mapped to W_j if and only if the x_{ij}-variables satisfy the system

$$\begin{cases} x_{ij} \geq 0 & i \in \{1, \ldots, p\},\ j \in \{1, \ldots, q\} \\ \sum_{j=1}^{q} x_{ij} = |U_i| & i \in \{1, \ldots, p\} \\ \sum_{i=1}^{p} x_{ij} \geq |W_j| & j \in \{1, \ldots, q\}. \end{cases} \tag{1}$$

The mapping f is a homomorphism from G to H if and only if the following holds: for each pair of variables $x_{ij}, x_{i'j'}$ such that $x_{ij} > 0$ and $x_{i'j'} > 0$, if U_i is adjacent to $U_{i'}$, then W_j is adjacent to $W_{j'}$.

We are now ready to give our algorithm. We first determine the set S and T. We then determine the U-sets and the W-sets. We guess a set R of indices (i, j) and only allow the variables x_{ij} for $(i, j) \in R$ to get non-zero value. Hence, we set $x_{ij} = 0$ for $(i, j) \notin R$. We then check whether for all pairs $(i, j), (i', j') \in R$, if U_i is adjacent to $U_{i'}$, then W_j is adjacent to $W_{j'}$. If not, then we discard R and guess a next one. Else we solve the system (1). If the system has an integer

solution, then the algorithm returns YES; otherwise we try a next guess of R. If all guesses fail, then the algorithm returns NO.

The correctness of the above algorithm follows from the aforementioned observations. We now estimate the running time. We can find S and T in time $1.2738^k n^{O(1)}$ and $1.2738^k m^{O(1)}$, respectively [3]. Then the sets $U_1, \ldots, U_p$ and $W_1, \ldots, W_q$ can be constructed in time $1.2738^k (nm)^{O(1)}$. The number of variables x_{ij} is $pq \leq (k + 2^k)^2 = 2^{O(k)}$. This means that there are at most $2^{2^{O(k)}}$ possibilities to choose R. By Lemma 1, system (1) (with some variables x_{ij} set to be zero) can be solved in time $2^{2^{O(k)}}(nm)^{O(1)}$. Hence, the total running time is $2^{2^{O(k)}}(nm)^{O(1)}$. This completes the proof of Theorem 3. □

5 Conclusions

Our complexity study shows that the SURJECTIVE HOMOMORPHISM problem is already NP-complete on a number of very elementary graph classes such as linear forests, trees of small pathwidth, unions of complete graphs, cographs, split graphs and proper interval graphs. We conclude that there is not much hope for finding tractable results in this direction, and consider the computational complexity classification of the SURJECTIVE $(-, H)$-HOMOMORPHISM problem as the main open problem; note that SURJECTIVE $(G, -)$-HOMOMORPHISM is trivially polynomial-time solvable for any guest graph G.

As we observed in Section 1, the SURJECTIVE $(-, H)$-HOMOMORPHISM problem is NP-complete already for any fixed host graph H that is nonbipartite. We also mentioned the existence of a bipartite graph H for which the problem is NP-complete [2] and that the problem can be solved in polynomial time whenever the host graph H is a fixed tree [14]. The paper of Feder et al. [7] on retractions provides a good starting point for the next step as we explain below.

A *pseudoforest* is a graph in which each connected component has at most one cycle. The RETRACTION problem is to test whether a graph G retracts to a graph H. Feder et al. [7] consider this problem for graphs that may have self-loops. Applying their result to simple graphs yields the following. For any pseudoforest H, the $(-, H)$-RETRACTION problem is NP-complete if H is nonbipartite or contains a cycle on at least 6 vertices, and it is polynomial-time solvable otherwise. It is an interesting open problem to show that $(-, H)$-RETRACTION and SURJECTIVE $(-, H)$-HOMOMORPHISM are polynomially equivalent for any fixed host graph H. All the evidence so far seems to suggest this.

References

1. Adiga, A., Chitnis, R., Saurabh, S.: Parameterized Algorithms for Boxicity. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010. LNCS, vol. 6506, pp. 366–377. Springer, Heidelberg (2010)
2. Bodirsky, M., Kára, J., Martin, B.: The complexity of surjective homomorphism problems – a survey (manuscript), ArXiv, http://arxiv.org/abs/1104.5257

3. 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)
4. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Applied Mathematics 101, 77–114 (2000)
5. Dalmau, V., Kolaitis, P.G., Vardi, M.Y.: Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 310–326. Springer, Heidelberg (2002)
6. Enciso, R., Fellows, M.R., Guo, J., Kanj, I., Rosamond, F., Suchý, O.: What Makes Equitable Connected Partition Easy. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 122–133. Springer, Heidelberg (2009)
7. Feder, T., Hell, P., Jonsson, P., Krokhin, A., Nordh, G.: Retractions to pseudoforests. SIAM Journal on Discrete Mathematics 24, 101–112 (2010)
8. 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)
9. Fiala, J., Kratochvíl, J.: Locally constrained graph homomorphisms – structure, complexity, and applications. Computer Science Review 2, 97–111 (2008)
10. Fiala, J., Golovach, P.A., Kratochvíl, J.: Parameterized complexity of coloring problems: Treewidth versus vertex cover. Theor. Comp. Sci. 412, 2513–2523 (2011)
11. Fiala, J., Paulusma, D.: A complete complexity classification of the role assignment problem. Theor. Comp. Sci. 349, 67–81 (2005)
12. Flum, J., Grohe, M.: Parameterized complexity theory. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin (2006)
13. Frank, A., Tardos, É.: An application of simultaneous Diophantine approximation in combinatorial optimization. Combinatorica 7, 49–65 (1987)
14. Golovach, P.A., Paulusma, D., Song, J.: Computing Vertex-Surjective Homomorphisms to Partially Reflexive Trees. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 261–274. Springer, Heidelberg (2011)
15. Garey, M.R., Johnson, D.R.: Computers and intractability. Freeman, NY (1979)
16. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. Journal of the ACM 54 (2007)
17. Hell, P., Nešetřil, J.: On the complexity of H-colouring. Journal of Combinatorial Theory, Series B 48, 92–110 (1990)
18. Hell, P., Nešetřil, J.: Graphs and homomorphisms. Oxford University Pr. (2004)
19. Lenstra Jr., H.W.: Integer programming with a fixed number of variables. Math. Oper. Res. 8, 538–548 (1983)
20. Martin, B., Paulusma, D.: The Computational Complexity of Disconnected Cut and 2K2-Partition. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 561–575. Springer, Heidelberg (2011)
21. Vikas, N.: Computational complexity of compaction to reflexive cycles. SIAM Journal on Computing 32, 253–280 (2002)
22. Vikas, N.: Compaction, retraction, and constraint satisfaction. SIAM Journal on Computing 33, 761–782 (2004)
23. Vikas, N.: A complete and equal computational complexity classification of compaction and retraction to all graphs with at most four vertices and some general results. J. Comput. Syst. Sci. 71, 406–439 (2005)
24. Vikas, N.: Algorithms for Partition of Some Class of Graphs under Compaction. In: Fu, B., Du, D.-Z. (eds.) COCOON 2011. LNCS, vol. 6842, pp. 319–330. Springer, Heidelberg (2011)

Broadcast Domination on Block Graphs in Linear Time*

Pinar Heggernes and Sigve H. Sæther

Department of Informatics, University of Bergen, Norway
{pinar.heggernes,sigve.sether}@ii.uib.no

Abstract. A broadcast domination on a graph assigns an integer value $f(u) \geq 0$ to each vertex u, such that every vertex u with $f(u) = 0$ is within distance $f(v)$ from a vertex v with $f(v) > 0$. The Broadcast Domination problem seeks to compute a broadcast domination where the sum of the assigned values is minimized. We show that Broadcast Domination can be solved in linear time on block graphs. For general graphs the best known algorithm runs in time $\mathcal{O}(n^6)$. For trees and interval graphs, linear-time algorithms are known. As block graphs form a superclass of trees, our result extends the classes of graphs on which this problem is solvable in linear time.

1 Introduction

Dominating Set and its variations are some of the most studied problems in graph theory; the number of papers on domination in graphs is in the thousands and there are several well known surveys and books on the topic, e.g., [7,8,9]. In 2002, Erwin [6] introduced the notion of *broadcast domination*. A *broadcast* on an undirected unweighted graph is a function from the vertices of the graph to non-negative integers. Such a function f is a *dominating broadcast* if each vertex u either has $f(u) > 0$ or is at distance at most $f(v)$ from another vertex v. The problem of computing a dominating broadcast that minimizes the sum of the function values of all the vertices is referred to as Broadcast Domination.

A few years after Erwin introduced the problem, it was discovered that it can be solved in polynomial time on all graphs [10]. However the running time of the algorithm for arbitrary graphs on n vertices is $\mathcal{O}(n^6)$. Algorithms for Broadcast Domination that run in time linear in the number of vertices and edges of the graph are known for interval graphs [2] and for trees [4]. Trees are also studied combinatorially with respect to their optimal dominating broadcast functions [3,11].

In this paper, we extend the results on trees by presenting a linear-time algorithm for Broadcast Domination on block graphs. Block graphs are the graphs in which each cycle is a clique, and they form thus a superclass of trees. Interestingly, our approach is substantially different and simpler than the one

* This work is supported by the Research Council of Norway.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 172–183, 2012.

used in the best known algorithm for trees [4]. However, the way to our algorithm goes through a number of new structural results on dominating broadcasts which we prove first.

2 Definitions and Notation

We work on undirected, unweighted, connected graphs, denoted by $G = (V, E)$. The set of vertices and the set of edges of a graph G are also denoted by $V(G)$ and $E(G)$, respectively. Throughout the paper we will use $n = |V(G)|$ and $m = |E(G)|$. The *distance* between two vertices u and v in G, denoted by $d_G(u, v)$, is the minimum number of edges on a path between u and v. The *eccentricity* of a vertex v, denoted by $e_G(v)$, is the largest distance from v to to any vertex of G. The *radius* of G, denoted by $rad(G)$, is the smallest eccentricity in G. The *diameter* of G, denoted by $diam(G)$, is the largest eccentricity in G. The *length* of a path is the number of edges it contains. A path is a *shortest path* in G if its length is equal to the distance between its end vertices. We define a *diametral path* in G to be a shortest path of length $diam(G)$.

Given a vertex v in G and a positive integer k, the *ball* $B_G(v, k)$ is the set $\{x \in V(G) \mid d_G(v, x) \leq k\}$. The *neighbourhood* of a vertex v in G is the set of all the vertices adjacent to v, denoted by $N_G(v)$. The *degree* of a vertex v is defined by $deg_G(v) = |N_G(v)|$. We also define $N_G[v] = N_G(v) \cup \{v\}$. The neighbourhood of a set of vertices S, denoted $N_G(S)$, is the set of all vertices adjacent to vertices of S, but which are not in S themselves. Thus, $N_G(B_G(v, k)) = B_G(v, k+1) \setminus B_G(v, k)$. If G is clear from context, we omit the subscript in these definitions. Two sets S_1 and S_2 of vertices of a graph G are *adjacent* if there exists a vertex $v_1 \in S_1$ and a vertex $v_2 \in S_2$ such that v_1 and v_2 are adjacent in G. A set of vertices $S \subset V(G)$ is a *separator* if $G[V(G) \setminus S]$ is disconnected. If a single vertex is a separator then it is called a *cut-vertex*. If P is a path then xPy refers to the subpath in P from x to y, i.e., x_3Px_5 is the path x_3, x_4, x_5 if $P = x_1, x_2, x_3, x_4, x_5, x_6, x_7$.

A function $f : V \to \{0, 1, \cdots, diam(G)\}$ is a *broadcast* on $G = (V, E)$. The set of *broadcast dominators* defined by f is the set $V_f = \{v \mid f(v) > 0\}$. A vertex u is *dominated* if there is a vertex $v \in V_f$ such that $d(u, v) \leq f(v)$. In that case we say that u is dominated by v. A broadcast on G is *dominating* if every vertex in $V(G)$ is dominated. In this case f is also called a *broadcast domination*. The *cost* of a dominating broadcast f on a subset $S \subseteq V$ is $\sigma_f(S) = \sum_{v \in S} f(v)$. We denote $\sigma_f(V)$ also as $\sigma_f(G)$, and refer to it as *the cost of* f *on* G. Note that there is always a dominating broadcast of cost $rad(G)$.

The *broadcast domination number*, $\gamma_b(G)$ is the smallest cost of a dominating broadcast on G, and the Broadcast Domination problem is to compute $\gamma_b(G)$. If $\sigma_f(G) = \gamma_b(G)$ then we call f an *optimal* broadcast on G. For a disconnected graph, an optimal dominating broadcast is the union of optimal dominating broadcasts of its connected components. This justifies that it is sufficient to study the problem on connected graphs. A dominating broadcast is *efficient* if every vertex is dominated by exactly one vertex. Dunbar et al. [5] proved that for any non-efficient dominating broadcast f on a graph $G = (V, E)$, there is an efficient

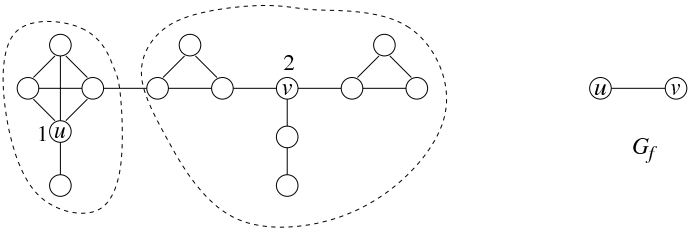


Fig. 1. A block graph G with a sparse broadcast f is shown on the left, where the broadcast dominators are u and v, with $f(u) = 1$ and $f(v) = 2$. The balls $B(u, 1)$ and $B(v, 2)$ are indicated on G. On the right G_f is shown.

dominating broadcast f' on G such that $|V_{f'}| < |V_f|$ and $\sigma_{f'}(G) = \sigma_f(G)$. An efficient dominating broadcast f on $G = (V, E)$ defines a *domination graph* $G_f = (V_f, E_f)$ such that there is an edge between two vertices $u, v \in V_f$ if and only if $B_G(u, f(u))$ is adjacent to $B_G(v, f(v))$ in G. See Fig. 1 for an example. Heggernes and Lokshtanov [10] proved that for any efficient dominating broadcast f on G, if G_f has a vertex of degree more than 2 then there is an efficient dominating broadcast f' on G such that $|V_{f'}| < |V_f|$ and $\sigma_{f'}(G) = \sigma_f(G)$.

We define a *sparse* broadcast on G to be an optimal broadcast f such that $|V_f|$ is minimized. As a consequence of the above, if a given optimal broadcast f on G is sparse then f is efficient and G_f is a path or a cycle.

3 General Properties of Dominating Broadcasts

We start by a simple observation that will allow us to replace the broadcast dominators of an induced subgraph with others, without affecting the rest of the broadcast dominators.

Observation 1. *Let $G = (V, E)$ be a graph, $A \subseteq V$, and $G' = G[A]$. If G has an optimal broadcast f such that $\bigcup_{a \in V_f \cap A} B(a, f(a)) = A$, and f' is an optimal broadcast on G', then G has an optimal broadcast f^* defined as follows:*

$$f^*(x) = \begin{cases} f'(x) & \text{if } x \in A, \\ f(x) & \text{if } x \in V \setminus A. \end{cases}$$

Proof. All the vertices that were dominated by vertices in $V_f \setminus A$ are dominated by the same vertices in V_{f^*}, and all the vertices that were dominated by vertices in A are dominated by vertices in $V_{f'}(G')$. So f^* is a dominating broadcast.

Since $\bigcup_{a \in V_f \cap A} B(a, f(a)) = A$, the restriction of f to A is a dominating broadcast on G'. Since f' is an optimal dominating broadcast on G', we have $\sum_{x \in A} f'(x) \leq \sum_{x \in A} f(x)$. Since the function values of the vertices outside of A did not change, we obtain that $\sigma_{f^*}(G) \leq \sigma_f(G)$ and hence f^* is an optimal dominating broadcast on G. □

The next lemmas and corollaries provide us with tools to decide whether a given dominating broadcast is sparse.

Lemma 1. *Let $B_1 = B(v, k_v)$ and $B_2 = B(u, k_u)$ be two balls in a graph G. If $k_v \leq k_u$, then there exists a vertex z in G such that $B_1 \cup B_2 \subseteq B(z, \ell)$ where*

$$\ell = \begin{cases} k_u & \text{if } B_1 \subseteq B_2, \\ \left\lceil \frac{d_G(u,v)+k_v+k_u}{2} \right\rceil & \text{otherwise.} \end{cases}$$

Proof. If B_1 is a subset of B_2, then let $z = u$ and observe that $B_1 \cup B_2 \subseteq B(z, \ell) = B_2$. Otherwise, let z be a vertex on a shortest path from u to v such that $d_G(u, z) = \lceil (d_G(u, v) + k_v - k_u)/2 \rceil$. Then $d_G(z, x) \leq k_u + d_G(z, u) = \ell$ for every vertex x in B_2, and hence $B_2 \subseteq B(z, \ell)$. For every vertex y in B_1, $d_G(z, y) \leq k_v + d_G(v, z) = k_v + d_G(v, u) - d_G(u, z) \leq \ell$. Consequently, also $B_1 \subseteq B(z, \ell)$. □

By excluding the cases from the above formula when $\left\lceil \frac{d_G(u,v)+k_v+k_u}{2} \right\rceil < k_u$, we can simplify the result to the following corollary.

Corollary 1. *Let $B_1 = B(v, k_v)$ and $B_2 = B(u, k_u)$ be two balls in a graph G. If $k_u + k_v \geq d_G(u, v) + 2k$ for an integer $k \leq \min\{k_u, k_v\}$, then there exists a vertex $z \in G$ such that $B_1 \cup B_2 \subseteq B(z,\ k_u + k_v - k)$.*

Lemma 2. *Let x, y, z be three vertices in a graph G. Let P_y be a shortest path from x to y in G, and let P_z be a shortest path from x to z in G. If $P_y \cap P_z$ contains more vertices than x, and the integers k_x, k_y, k_z are such that $B(x, k_x)$ is adjacent to both $B(y, k_y)$ and $B(z, k_z)$, then there exists a vertex v in G such that $B(x, k_x) \cup B(y, k_y) \cup B(z, k_z) \subseteq B(v, k_x + k_y + k_z)$.*

Proof. If there exists any vertex other than x in $P_y \cap P_z$, then any shortest path from x to that other vertex is a subpath of a shortest path to both y and z from x. Consequently, the vertex u adjacent to x in P_y must be 1 closer to both z and y than x is. Since u and x are adjacent, we notice that $B(x, k_x) \subseteq B(u, k_x + 2)$. We observe that $B(u, k_x + 2)$ overlaps $B(z, k_z)$ in such a way that we can apply Corollary 1. Therefore, there exists a vertex w such that $B(u, k_x + 2) \cup B(z, k_z) \subseteq B(w, k_z + k_x + 1)$. And this ball overlaps $B(y, k_y)$ in such a way that, again by Corollary 1, there exists a vertex v such that $B(w, k_z + k_x + 1) \cup B(y, k_y) \subseteq B(v, k_z + k_x + k_y)$. Hence,

$$\begin{aligned} B(x, k_x) \cup B(y, k_y) \cup B(z, k_z) &\subseteq B(y, k_y) \cup B(u, k_x + 2) \cup B(z, k_z) \\ &\subseteq B(y, k_y) \cup B(w, k_z + k_x + 1) \\ &\subseteq B(v, k_z + k_x + k_y). \end{aligned}$$ □

Corollary 2. *Let f be a sparse broadcast on a graph $G = (V, E)$, and let x, y and z be distinct vertices in V_f such that y and z are adjacent to x in G_f. Then the intersection of a shortest path from x to y in G and a shortest path from x to z in G is exactly the set $\{x\}$.*

4 Structural Properties of Dominating Broadcasts on Block Graphs

A graph is a *block graph* if the vertices of every cycle form a clique. The following theorem will be used in our proofs.

Theorem 1 ([1,12]). *Given four vertices x, y, z and w in a block graph G, the two largest of the following three sums are equal.*

1. $d(x, y) + d(z, w)$
2. $d(x, z) + d(y, w)$
3. $d(x, w) + d(y, z)$

Since every cycle in a block graph is a clique and any shortest path contains at most two vertices of a clique, it follows that a shortest path between any pair of vertices is unique in a block graph. In particular, in a block graph, every vertex on a shortest path between two vertices s and t is a cut-vertex, separating s from t.

Lemma 3. *If G is a block graph then $rad(G) = \lceil diam(G)/2 \rceil$.*

Proof. If G is a complete graph then the statement trivially follows. Assume that G is not complete and let $P = s, \ldots, t$ be a diametral path in G. Then there is a vertex $x \in P$ such that $\min\{d(s, x), d(t, x)\} = \lfloor diam(G)/2 \rfloor$. Recall that x is a cut-vertex. We show that $e(x) \leq \lceil diam(G)/2 \rceil$. Assume for contradiction that there is a vertex y such that $d(x, y) > \lceil diam(G)/2 \rceil$. Since x disconnects s and t, x must also disconnect y from either s or t. Without loss of generality, let x disconnect y and t. In this case, $d(y, t) = d(y, x) + d(x, t)$. Since $d(y, x) > \lceil diam(G)/2 \rceil$ and $d(t, x) \geq \lfloor diam(G)/2 \rfloor$, we reach the contradiction that $d(y, t) > diam(G)$. Hence we conclude that $e(x) \leq \lceil diam(G)/2 \rceil$. Consequently $rad(G) \leq \lceil diam(G)/2 \rceil$. Since $rad(G) \geq \lceil diam(G)/2 \rceil$, the radius must be exactly $\lceil diam(G)/2 \rceil$. □

Lemma 4. *Let G be a block graph and s be a vertex of G. Then $e(s) = diam(G)$ if and only if G has a vertex x such that $d(x, s) \geq d(x, y)$ for all $y \in V(G)$.*

Proof. One direction is trivial as we can take x to be the end vertex of the diametral path starting from s. For the other direction, let x be a vertex such that $d(x, s) \geq d(x, y)$ for all $y \in V(G)$. Let $P = a, \ldots, b$ be a diametral path in G, and let $d_1 = d(a, b) + d(x, s)$, $d_2 = d(a, x) + d(b, s)$, $d_3 = d(a, s) + d(b, x)$. Since $d(x, s)$ is greater than or equal to both $d(a, x)$ and $d(b, x)$, and since $d(a, b)$ is greater than or equal to every distance in G, we see that the sum d_1 can be no less than either of d_2 or d_3. Without loss of generality, let $d_2 \leq d_3$. By Theorem 1 we know $d_1 = d_3$. Since $d(x, s) \geq d(x, b)$ we have that $d(a, b) \leq d(a, s)$. And since $d(a, b) = diam(G)$, we also have that $d(s, a) = diam(G)$, and hence $e(s) = diam(G)$. □

Observation 2. *Let G be a block graph and let f be a sparse broadcast on G. Then G_f is a path.*

Proof. As mentioned in the introduction, G_f is either a path or a cycle. Assume for contradiction that G_f is a cycle. Since f is efficient, V_f is an independent set in G. However, in combination with Corollary 2, this means we must have an induced cycle of length at least 6 in G, which contradicts that G is a block graph. □

Lemma 5. *Let G be a block graph and let f be a sparse broadcast on G. For every clique C of size at least 3, all vertices in C are dominated by the same broadcast dominator in V_f.*

Proof. Since G_f is a path and f is efficient by the above, the vertices of C can be dominated by at most two distinct vertices. C contains more than two vertices, therefore at least two vertices x and y in C must be dominated by the same vertex z in V_f. Since the distances in G between the vertices in C are all 1, we have by Theorem 1 that $d_G(w, z) \leq d_G(x, z) = d_G(y, z)$ for every vertex $w \in C$. Therefore, every vertex w in C is dominated by z. □

5 Algorithmic Properties of Dominating Broadcasts on Block Graphs

Recall that any clique intersects with a diametral path in at most two vertices. Given a diametral path P, we define $C(P)$ to be the union of all cliques that intersect with P in exactly two vertices. A set $C \subseteq V(G)$ is called a *core* of G if $C = C(P)$ for a diametral path P in G. Through the series of lemmas of this section we will prove that in every block graph $G = (V, E)$ there is an optimal broadcast f and a core C, such that every dominator in V_f belongs to C. Finally Lemma 11 will enable us to find the dominators and their respective weights in f, one by one, as will be described in the resulting algorithm in the next section. We start by defining two operations. These operations are illustrated in Fig. 2.

Definition 1. *Given a ball $B = B(a_0, k)$ and a path $P = a_0, a_1, \ldots, a_p$ in a graph G, and a positive integer $l \leq p$, an* increase of l vertices along P for B *is the ball $B(a_l, k + l)$. This will also be referred to as the ball* $\textsc{Increase}(B, P, l)$.

Definition 2. *Given a ball $B = B(a_0, k)$ and a path $P = a_0, a_1, \ldots, a_p$ in a graph G, and a positive integer $l \leq \min\{k - 1, p\}$, a* decrease of l vertices along P for B *is the ball $B(a_l, k - l)$. This will also be referred to as the ball* $\textsc{Decrease}(B, P, l)$.

For all graphs we can make the following observations, the first of which is immediate.

Observation 3. *Given a ball B_1 in a graph, if there is a path aPb such that $B_2 = \textsc{Increase}(B_1, aPb, l)$, then $B_1 = \textsc{Decrease}(B_2, bPa, l)$.*

Observation 4. *Given a ball $B = B(v, k)$ in a graph G, a path P of length ℓ in G from v to any vertex, and a positive integer $l \leq \ell$, $B \subseteq \textsc{Increase}(B, P, l)$.*

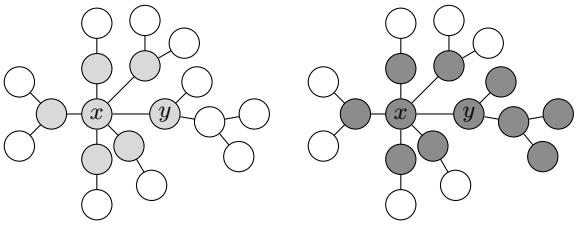


Fig. 2. A ball $B_1 = B(x, 1)$ in green to the left, and the ball $\textsc{Increase}(B_1, (x, y), 1) = B(y, 2)$ in blue to the right in the same graph

Proof. Let $B(v', k') = \textsc{Increase}(B, P, l)$. Since $d_G(v, v') \leq l$, we have that $d_G(v', x) \leq d_G(v', v) + d_G(v, x) \leq l + k = k'$, for every $x \in B$. Thus every vertex in B is also in $B(v', k')$. □

Lemma 6. *Let P be a shortest path between two vertices s and t in a block graph G. Let u be a vertex of P and let k_1 and k_2 be two positive integers such that $k_1 < k_2$ and $k_1 \leq d_G(u, s)$. Given the balls $B_1 = B(u, k_2)$ and $B_2 = \textsc{Decrease}(B_1, uPs, k_1)$, if $t \in B_2$ then $B_1 \cap B(s, d(s, t)) = B_2 \cap B(s, d(s, t))$.*

Proof. Let u' be the vertex on uPs such that $B(u', k_2 - k_1) = B_2$. It should be clear that uPs is the shortest path from u to s. Recall that in this case u' is a cut-vertex separating u and s. Thus, there is a set of vertices $S \subseteq B(s, d(s, t))$ such that every path from u to a vertex in S goes through u', and every path from s to a vertex in $T = B(s, d(s, t)) \setminus S$ goes through u'. The path uPu' is a shortest path since it is a sub path of P. Hence, $d(u, u') = k_1$. For every vertex $v \in S$, we thus have $d(u', v) = d(u, v) - k_1$. Consequently, every vertex in $B_1 \cap S$ is at distance at most $k_2 - k_1$ from u'. Hence $B_1 \cap S \subseteq B_2$.

Recall that u' is on every path from s to a vertex of T. Since every vertex $v \in T$ is at distance at most $d(s, t)$ from s, we get the following: $d(v, u') + d(u', s) = d(v, s) \leq d(s, t) = d(s, u') + d(u', t)$. Thus, each vertex in T is within distance $d(u', t) \leq k_2 - k_1$ from u'. Hence $T \subseteq B_2$. Hence $B_1 \cap B(s, d(s, t)) = B_2 \cap B(s, d(s, t))$. □

Lemma 7. *Let G be a block graph, let $P = s_o, \ldots, s_k, \ldots, s_p$ be a diametral path in G, and let $v \neq s_k$ be a vertex of G. If $2k \leq p$ and $s_0 \in B(v, k)$ then $B(v, k) \subset B(s_k, k)$.*

Proof. Since $s_0 \in B(v, k)$, we see that $d(s_0, x) \leq 2k$ for any vertex $x \in B(v, k)$. Hence, $B(v, k) \subseteq B(s_0, 2k)$. Note that the ball $B(s_k, k)$ is the same as the ball $\textsc{Decrease}(B(s_0, 2k), P, k)$. Since $B(s_p, p) = V(G)$, by Lemma 6 we get that $B(s_k, k) = B(s_0, 2k)$. We have thus proved that $B(v, k) \subseteq B(s_0, 2k) = B(s_k, k)$. However, since s_k is on the shortest path from s_0 to s_{2k} (the path $s_0 P s_{2k}$), s_k is a cut-vertex separating the two vertices. That means s_k is on the shortest path from v to either s_0 or s_{2k}. Since $d(v, s_0) \leq d(s_k, s_0)$, s_k must be on the path from v to s_{2k}. The distance from v to s_{2k} must therefore be larger than $d(s_k, s_{2k}) = k$. Consequently, $B(s_k, k) \neq B(v, k)$. Combining this with the above, we get that $B(v, k) \subset B(s_k, k)$. □

Lemma 8. *Let G be a block graph, and let P be a diametral path $s_0, \ldots, s_i, \ldots, s_p$. If f is a sparse broadcast on G, and $|V_f| > 1$, then s_0 is dominated by a vertex s_k in P such that $f(s_k) = k$.*

Proof. Let s' be the vertex dominating s_0. Since $|V_f| > 1$ and f is optimal, the weight of every dominator must be less than $rad(G)$. So, $f(s') \leq rad(G) - 1 = \lceil p/2 \rceil - 1$, by Lemma 3. We have by Lemma 7 that, unless s' is in P, there exists a vertex x such that $B(s', f(s')) \subset B(x, f(s'))$. However, in that case there must exist a non-efficient optimal broadcast g such that $|V_g| = |V_f|$. However, this makes g a sparse broadcast on G, and thus g must be efficient as argued in the preliminaries. This is a contradiction, and hence $s' = s_k$ for some k, and $f(s_k) = k$. □

Lemma 9. *Let G be a block graph and let C be a core of G. If f is a sparse broadcast on G such that $|V_f| > 1$, then each vertex in C is dominated by a dominator in C.*

Proof. Let $P = s_0, s_1 \ldots, s_p$ be a diametral path such that $C = C(P)$. Note that if each vertex on P is dominated by a vertex in C, then each vertex in $C \setminus P$ must also be dominated by a vertex in C by Lemma 5. By Lemma 8, vertices s_0 and s_p are dominated by vertices in $P \subseteq C$. Let s'_0 and s'_p be the vertices dominating s_0 and s_p, respectively. Let $s_i, s_{i+1}, \ldots, s_j$ be the vertices on P not dominated by s'_0 or s'_p and let x be the vertex dominating a vertex $s_d \in s_i P s_j$. Since either of s_i and s_j (alone) separates s'_0 and s'_p, the dominators of the vertices in $s_i P s_j$ must be between s'_0 and s'_p in G_f. Therefore, the degree of x in G_f must be exactly 2. Let x_l and x_r be the vertices in G_f to the left and right of x, respectively. Notice that C forms a union of some maximal cliques. By the definition of block graphs, each component of $G - C$ must therefore be adjacent to only one vertex in C. So, if x is not in C, then there must exist a single vertex $u \in C$ that disconnects x from C in G. However, then the shortest path from x to both x_l and x_r must contain u. By Lemma 2, this implies that u and x must be the same vertex, a contradiction. Therefore, if x is dominating a vertex in P, x itself must be in C. □

Lemma 10. *Let G be a block graph and let C be a core of G. If f is a sparse broadcast on G and $|V_f| > 1$, then $V_f \subseteq C$.*

Proof. Recall that since f is sparse, it is efficient and G_f is a path. Let $P = s_0, s_1, \ldots, s_p$ be a diametral path in G, such that $C = C(P)$. Let s'_0 and s'_p be the vertices dominating s_0 and s_p, respectively. By Lemma 8, s'_0 and s'_p are on P, and by Lemma 9, the vertices between s'_0 and s'_p in G_f are in C. Therefore, if $V_f \not\subseteq C$, there must exist a vertex $x \in V_f \setminus C$ which is adjacent to s'_0 or s'_p in G_f. Without loss of generality, let x be adjacent to s'_0 in G_f. Let y be the vertex adjacent to s'_0 in G on the path $s'_0 P s_p$. We have two possibilities: $d_G(y, x) < d_G(s'_0, x)$ or $d_G(y, x) \geq d_G(s'_0, x)$.

In the first case y separates s'_0 and x in G. However, since y also separates in G s'_0 and the other neighbour of s'_0 in G_f, we know by Lemma 2 that f is not sparse, giving a contradiction. In the second case, y separates x and s_p. Note that $d_G(s_p, y) = d_G(s_p, s'_0) - 1$. Since $d_G(s'_0, s_0) \leq f(s'_0)$ and f is efficient, with $f(x) > 0$, we have $d_G(s'_0, x) = f(s'_0) + f(x) + 1 > d_G(s'_0, s_0) + 1$. This leads to the following contradiction: $d_G(s_p, x) = d_G(s_p, y) + d_G(y, x) > d_G(s_p, s'_0) + d_G(s'_0, s_0) = diam(G)$.

Consequently, there can be no vertex x in $V_f \setminus C$ if f is sparse. □

Lemma 11. *Let $P = s_0, s_1, \ldots, s_k, \ldots, s_p$ be a diametral path in a block graph G, let f be a sparse broadcast where $|V_f| > 1$, and assume that all vertices in $B(s_k, k)$ are dominated by the same dominator in V_f for $k \leq p/2$.*

If both of the following statements are true then there exists an optimal broadcast f' on G such that $f'(s_k) = k$, and if either of them is false then all vertices in $B(s_{k+1}, k+1)$ have the same dominator in V_f.

1. $d(s_0, x) = 2k + 1 \Rightarrow x = s_{2k+1}$
2. $d(s_p, x) = d(s_p, s_{2k}) + 1 \Rightarrow x \in N(s_{2k})$

Proof. Let s_i be the vertex that dominates $B(s_k, k)$. Since f is sparse, $f(s_i) = i$. Consequently, $i \geq k$. By Lemma 5, if s_{2k} and s_{2k+1} have a neighbour in common, they must have the same dominator, which implies that $i > k$, since $s_{2k+1} \notin B(s_k, k)$.

Assume that (1) is false, i.e., there is a vertex x such that $d(s_0, x) = 2k + 1$ and $x \neq s_{2k+1}$. If x is a neighbour of s_{2k+1} and s_{2k}, then $i > k$ as argued above, and hence all vertices in $B(s_{k+1}, k+1)$ are dominated by s_i. Assume that x is not adjacent to both of them. Hence s_{2k} separates s_{2k+1} and x, and $i = k$. But then the degree of s_i in G_f is larger than 1, which contradicts that G_f is a path.

Assume that (2) is false, i.e., there is a vertex x such that $d(s_p, x) = d(s_p, s_{2k}) + 1$ and $x \notin N(s_{2k})$. Let $x' \in C(P)$ be the vertex dominating x in f. We have $d(s_p, x') + d(x', x) \geq d(s_p, x) = d(s_p, s_{2k}) + 1 \geq d(s_p, x') + d(x', s_{2k})$. Hence, s_{2k} is dominated by x'. However, this is also true for s_{2k+1}, so $i > k$, as above.

Assume that both (1) and (2) are true. If $i = k$ then we are done. Assume that $i > k$. Let $R = B(s_p, d(s_p, s_{2k}))$ and let $B = B(s_i, i)$. Note that $d(s_{2k}, s_{i+k}) = i - k$, and thus, $s_{2k} \in B' = \textsc{Decrease}(B, s_i P s_p, k)$. By Lemma 6, this implies that $B' \cap R = B \cap R$. Since $B \setminus R \subset B(s_k, k)$, we now see that $B = B' \cup B(s_k, k)$. We can construct a new broadcast f' as follows:

$$f'(x) = \begin{cases} k & \text{if } x = s_k \\ 0 & \text{if } x = s_i \\ i - k & \text{if } x = s_{i+k} \\ f(x) & \text{otherwise} \end{cases}$$

By the above arguments, f' dominates the same vertices as f, and clearly $\sigma_{f'}(G) = \sigma_f(G)$. (Observe that now s_{2k} is dominated by both s_k and s_{i-k}, so f' is not efficient.) □

The above result can be used to construct a simple greedy algorithm for calculating an optimal broadcast domination on block graphs:

As long as $\gamma_b(G) < rad(G)$ there will always be a sparse broadcast f such that $|V_f| > 1$, and a diametral path $s_0, s_1, \ldots, s_p$ such that every vertex in $B(s_1, 1)$ are dominated by the same vertex. That creates the basis for Lemma 11. An algorithm can thus iterate through the possible values of k until both statements (1) and (2) of Lemma 11 are satisfied. When this happens, simply remove the dominated vertices $B(s_k, k)$, and repeat. Even though the graph is shrinking for each iteration, we remove non-overlapping induced subgraphs, and by Observation 1, all the individual optimal broadcast dominations for the subgraphs add up to an optimal dominating broadcast for the entire graph.

By the proof of Lemma 3, finding the optimal broadcast for G' when $\gamma_b(G') = rad(G)$, is as simple as picking the middle vertex in a diametral path. The knowledge that $\gamma_b(G') = rad(G)$ is obtained when the algorithm reaches a k such that $B(s_k, k) = V(G')$.

6 A Linear-time Algorithm for Optimal Broadcasts on Block Graphs

We have now all the results we need to present our algorithm.

Theorem 2. Broadcast Domination *can be solved in linear time on block graphs.*

Proof. As explained at the end of the previous section, the algorithm finds the leftmost dominator in an efficient optimal broadcast for G, according to Lemma 11. It then removes all the vertices dominated by this vertex, and finds the leftmost dominator in an efficient optimal broadcast for the remaining subgraph. This is repeated until there are no vertices left, and thus every vertex is dominated. By Observation 1 and Lemma 11, the resulting broadcast is optimal.

The algorithm starts by finding a vertex t such that $e(t) = diam(G)$, using Lemma 4. A diametral path between t and a vertex s is used in the algorithm. However, we do not need to compute every vertex in the path for each iteration. We can calculate each vertex s_i on the path $s = s_0, s_1, \cdots, s_i, \cdots, s_p = t$ as we need it without knowing the rest of the path: The vertex s_i is equivalent to the vertex x such that $d_G(s_0, x) = i$ and $d_G(x, s_p) = p - i$.

We need not change t for each iteration either: each time the graph G changes into a smaller graph G', every vertex to the left of a vertex s_j in the old diametral path are removed. However, since $diam(G) = d(s, t) = d(s, s_j) + d(s_j, t)$ and the shortest path in G from s to any vertex in G' goes through s_j, no vertex in G' can be further away from s_j than t, as that would imply $d_G(s, t) < diam(G)$. Therefore, $e_{G'}(t) = diam(G')$, by Lemma 4.

Algorithm Block Graph Broadcast

Input: A block graph $G = (V, E)$
Output: An optimal broadcast f on G

for every v **in** V **do**
  $f(v) \leftarrow 0$
end for
$x \leftarrow$ an arbitrary vertex in V
$t \leftarrow$ a vertex at maximum distance from x
for $i \leftarrow 0$ **to** $|V| - 1$ **do**
  $T[i] \leftarrow$ the set of vertices at distance i from t
  for $v \in T[i]$ **do**
    $D[v] \leftarrow i$
  end for
end for
while $V \neq \emptyset$ **do**
  $s \leftarrow$ a vertex at maximum distance from t in G
  $S[0] \leftarrow \{s\}$
  $S[1] \leftarrow N(s)$
  $S[2] \leftarrow N(S[1]) \setminus \{s\}$
  $k \leftarrow 1$
  while $2k < D[s]$ **do**
    $S[2k+1] \leftarrow$ the vertices at distance $2k+1$ from s
    $S[2k+2] \leftarrow$ the vertices at distance $2k+2$ from s
    $s_{2k} \leftarrow$ the vertex in $S[2k]$ such that $D[s_{2k}] = D[s] - 2k$
    if $|S[2k+1]| = 1$ **and** $T[D[s] - 2k + 1] \subset N(s_{2k})$ **then break**
    $k \leftarrow k + 1$
  end while
  $v \leftarrow$ the vertex in $S[k]$ such that $D[v] = D[s] - k$, or s if $V = \{s\}$
  $f(v) \leftarrow k$
  $V \leftarrow V \setminus B(v, f(v))$
  $G \leftarrow G[V]$
end while
return f

The condition of the inner *while*-loop is for the base case when there is only one dominator left.

The running time of the algorithm is $\mathcal{O}(n+m)$: we start with a loop of $\mathcal{O}(n)$ iterations, initiating the values of f, and then populate T and D in $\mathcal{O}(n+m)$ by a breadth first search. The main loop might look like a bottle neck because of the time needed to populate entries of S in each iteration. However, each entry

$S[i]$ can be calculated in time proportional to the cardinality of $N[S[i-1]]$. Therefore, since each vertex appears in one and only one entry, the total running time to calculate all sets in S, is no more than $\mathcal{O}(n+m)$. Hence, the total running time of the entire algorithm is $\mathcal{O}(n+m)$. □

7 Concluding Remarks

A graph is *chordal* if it does not contain an induced cycle of length 4 or more. Although block graphs and interval graphs are unrelated to each other, they are both subclasses of chordal graphs. It is easy to see that every chordal graph G has an optimal broadcast domination f such that G_f is a path. Following the results of [10], this immediately gives an $\mathcal{O}(n^4)$-time algorithm for Broadcast Domination on chordal graphs. Is there a linear-time or an $\mathcal{O}(n^2)$-time algorithm for Broadcast Domination on chordal graphs?

We believe that Broadcast Domination can be solved in general in $\mathcal{O}(n^5)$ time. Is there a faster algorithm for Broadcast Domination on arbitrary graphs?

References

1. Bandelt, H.J., Mulder, H.M.: Distance-hereditary graphs. J. Comb. Theor., Ser. B 41, 182–208 (1986)
2. Chang, R.Y., Peng, S.L.: A linear-time algorithm for broadcast domination problem on interval graphs. In: Proceedings of the 27th Workshop on Combinatorial Mathematics and Computation Theory, pp. 184–188. Providence University, Taichung (2010)
3. Cockayne, E.J., Herke, S., Mynhardt, C.M.: Broadcasts and domination in trees. Disc. Math. 311, 1235–1246 (2011)
4. Dabney, J., Dean, B.C., Hedetniemi, S.T.: A linear-time algorithm for broadcast domination in a tree. Networks 53(2), 160–169 (2009)
5. Dunbar, J.E., Erwin, D.J., Haynes, T.W., Hedetniemi, S.M., Hedetniemi, S.T.: Broadcasts in graphs. Disc. Appl. Math. 154(1), 59–75 (2006)
6. Erwin, D.J.: Dominating broadcasts in graphs. Bull. Inst. Comb. Appl. 42, 89–105 (2004)
7. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Fundamentals of Domination in Graphs. Marcel Dekker (1998)
8. Haynes, T.W., Hedetniemi, S.T., Slater, P.J. (eds.): Domination in Graphs: Advanced Topics. Marcel Dekker (1998)
9. Hedetniemi, S.T., Laskar, R.C. (eds.): Topics on domination. North Holland (1990)
10. Heggernes, P., Lokshtanov, D.: Optimal broadcast domination of arbitrary graphs in polynomial time. Disc. Math. 306(24), 3267–3280 (2006)
11. Herke, S., Mynhardt, C.M.: Radial trees. Disc. Math. 309, 5950–5962 (2009)
12. Howorka, E.: On metric properties of certain clique graphs. J. Comb. Theor., Ser. B 27(1), 67–74 (1979)

Characterizing Certain Topological Specifications*

Bernhard Heinemann

Faculty of Mathematics and Computer Science,
University of Hagen,
58084 Hagen, Germany
bernhard.heinemann@fernuni-hagen.de

Abstract. We prove a characterization theorem à la van Benthem for a particular modal system called topologic, which is, among other things, suitable for specifying the interrelation between knowledge and topology. The comparison language arising naturally from the relevant semantics is well-known from the beginnings of topological model theory, and subset space bisimulations provide for the proper notion of invariance of formulas here.

1 Introduction

As is known, modal languages constitute a widespread formalism for the purposes of system specification, e.g., in the case of distributed or multi-agent scenarios. But once a specification language is to hand, trying to get to the bottom of its scope is a fundamental issue ever. The classical *van Benthem Characterization Theorem* gives a satisfactory answer to this problem in the case of basic modal logic: the bisimulation-invariant fragment of first-order logic and the entirety of those properties that can be specified by modal formulas coincide; see [4], Theorem 2.68. In this paper, we prove an analogous result for Moss and Parikh's bimodal language for topological spaces, $\mathcal{L}$, originating from [12] (see also [7]).

Let us take a quick look at the semantics of $\mathcal{L}$. Its basic units are composed of two ingredients, to wit, the actual state x of the world and an open neighborhood U of x. In knowledge-theoretic contexts, which serve us as an example here, U may be viewed as the current *epistemic state* of an agent under discussion, i.e., the set of those states that cannot be distinguished by what the agent topically knows. The two modalities of the language, K and $\Box$, quantify across all elements of U and 'downward' over all open sets contained in U, respectively. Thus K captures the common notion of *knowledge* (see [8]), and $\Box$ reflects *effort to acquire knowledge* since gaining knowledge goes hand in hand with a shrinkage of the epistemic state. In fact, knowledge acquisition is reminiscent of a topological procedure in this way. The appropriate logic for topological spaces, exactly topologic, was first determined by Georgatos in his thesis [10]. Meanwhile, a considerable amount of work was involved in the development of a corresponding

* Dedicated to Professor Rohit Parikh on the occasion of his 75th birthday.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 184–195, 2012.

theory; see Ch. 6 of the handbook [2] for a guide to the literature up to 2006. Summing all this and the more recent achievements up, one may state that the emerging system embodies a well-established toolkit for specification tasks related to reasoning about knowledge as well as topological reasoning, with the two-valued semantics involving both states and ambient open sets making up its distinctive feature.

Now is the time to explore the expressive power of the language $\mathcal{L}$. A careful analysis of its semantics will direct us to our goal, viz a preferably comprehensive description of $\mathcal{L}$-definable properties. To this end, we shall first identify the right comparison language for $\mathcal{L}$. It shows that this language coincides with the language L_2 for so-called *weak structures,* introduced by Flum and Ziegler in the monograph [9]. See that L_2 is a first-order language in essence, the machinery of first-order model theory (see, e.g., [6]) is applicable to our problem, and we shall make an extensive use of this in due course.

The very topic of Flum and Ziegler's book is a sublanguage of L_2 where quantification over set variables is restricted in a certain manner. This language, L_t, has proven a perfect match for the classic topological interpretation of usual (mono-)modal logic (see [1]), as the recent paper [5] shows. Unfortunately, we cannot exploit the expressivity issues obtained in that paper for our purposes, since the translations of $\mathcal{L}$-formulas do not generally satisfy those syntactic restrictions. The prize we have to pay for obtaining the desired characterization by means of the contemplated methods nevertheless, is to admit arbitrary subset spaces as semantic structures, as it is the case with the original Moss-Parikh logic.

The rest of this paper is organized as follows. In the next section, we supply the basic definitions from [7] needed subsequently, and we define a *standard translation* of the set of all $\mathcal{L}$-formulas which is induced by the semantics of $\mathcal{L}$. Finally in this section, we reason about the image of that translation. In Section 3, the concept of *saturated models* is revisited. Moreover, the idea of *modal saturation* that is relevant to our setting is investigated. Section 4 is devoted to *bisimulations,* in fact, *subset space bisimulations* introduced in [3]. The desired characterization theorem is then proved in Section 5. At the end of the paper, we add some concluding remarks.

2 The Languages We Consider

In this section, we first fix the language $\mathcal{L}$ underlying topologic and extract a translation into first-order logic from that. Afterwards, we recall the language L_2 before connecting $\mathcal{L}$ to it.

To begin with, we define the syntax of $\mathcal{L}$. Let $\mathsf{Prop} = \{p, q, \ldots\}$ be a denumerably infinite set of symbols called *proposition variables* (which should represent the basic facts about the states of the world). Then, the set Form of all $\mathcal{L}$*-formulas* over Prop is defined by the rule $\alpha ::= \top \mid p \mid \neg\alpha \mid \alpha \wedge \alpha \mid \mathsf{K}\alpha \mid \Box\alpha$. Later on, the boolean connectives that are missing here are treated as abbreviations, as needed. The dual operators of K and $\Box$ are denoted by L and $\Diamond$, respectively; K is called the *knowledge operator* and $\Box$ the *effort operator.*

We now turn to the semantics of $\mathcal{L}$. For a start, we define the relevant domains. We let $\mathcal{P}(X)$ designate the powerset of a given set X.

Definition 1 (Semantic Domains).

1. *Let X be a non-empty set (of* states*) and $\mathcal{O} \subseteq \mathcal{P}(X)$ a set of subsets of X. Then, the pair $\mathcal{S} = (X, \mathcal{O})$ is called a* subset frame.
2. *Let $\mathcal{S} = (X, \mathcal{O})$ be a subset frame. The set $\mathcal{N}_\mathcal{S} := \{(x, U) \mid x \in U \text{ and } U \in \mathcal{O}\}$ is called the* set of neighborhood situations *of $\mathcal{S}$.*
3. *Let $\mathcal{S} = (X, \mathcal{O})$ be a subset frame. An $\mathcal{S}$-*valuation *is a mapping $V : \mathsf{Prop} \to \mathcal{P}(X)$.*
4. *Let $\mathcal{S} = (X, \mathcal{O})$ be a subset frame and V an $\mathcal{S}$-valuation. Then, $\mathcal{M} := (X, \mathcal{O}, V)$ is called a* subset space (based on $\mathcal{S}$).

The term 'neighborhood situation' has been introduced just to denominate the semantic atoms of our modal language. Note that the first component of such a situation indicates the actual state of the world while the second indicates the uncertainty of the respective agent about it. Note also that $\mathcal{S}$-valuations only depend on states (and are independent of open sets thus). This is in accordance with the common practice of modelling; cf. [8].

Now, let a subset space $\mathcal{M}$ be given. We define the relation of satisfaction, $\models_\mathcal{M}$, between neighborhood situations of the underlying frame and $\mathcal{L}$-formulas from Form. In the following, neighborhood situations are often written without parentheses.

Definition 2 (Satisfaction and Validity). *Let $\mathcal{M} = (X, \mathcal{O}, V)$ be a subset space based on $\mathcal{S} = (X, \mathcal{O})$, and let $x, U \in \mathcal{N}_\mathcal{S}$ be a neighborhood situation of $\mathcal{S}$. Then*

$$
\begin{aligned}
x, U &\models_\mathcal{M} \top && && \text{is always true} \\
x, U &\models_\mathcal{M} p && :\Longleftrightarrow\ && x \in V(p) \\
x, U &\models_\mathcal{M} \neg\alpha && :\Longleftrightarrow\ && x, U \not\models_\mathcal{M} \alpha \\
x, U &\models_\mathcal{M} \alpha \wedge \beta && :\Longleftrightarrow\ && x, U \models_\mathcal{M} \alpha \text{ and } x, U \models_\mathcal{M} \beta \\
x, U &\models_\mathcal{M} \mathsf{K}\alpha && :\Longleftrightarrow\ && \forall\, y \in U : y, U \models_\mathcal{M} \alpha \\
x, U &\models_\mathcal{M} \Box\alpha && :\Longleftrightarrow\ && \forall\, U' \in \mathcal{O} : (x \in U' \subseteq U \Rightarrow x, U' \models_\mathcal{M} \alpha)\,,
\end{aligned}
$$

where $p \in \mathsf{Prop}$ and $\alpha, \beta \in \mathsf{Form}$. In case $x, U \models_\mathcal{M} \alpha$ is true we say that α holds in $\mathcal{M}$ at *the neighborhood situation x, U. Furthermore, an $\mathcal{L}$-formula α is called* valid in $\mathcal{M}$ *iff it holds in $\mathcal{M}$ at every neighborhood situation of $\mathcal{S}$.*

Note that the idea of knowledge and effort described in the introduction is made precise by this definition. In particular, knowledge *is defined* as validity at all states that are indistinguishable to the agent; cf. [8].

The defining clauses on the right hand side of the double arrows in Definition 2, give rise to a translation of $\mathcal{L}$-formulas into a two-sorted language of predicate logic. The target language should obviously contain state variables $v, v', \ldots$, set variables $\Upsilon, \Upsilon', \ldots$, a binary relation symbol ε of the sort $(state, set)$, and a binary

relation symbol $\sqsubseteq$ of the sort (set, set); moreover, a unary predicate symbol P_p of the sort $(state)$ is associated with every proposition variable $p \in \mathsf{Prop}$. Then, after fixing a state variable v, p is translated to $P_p(v)$, and this assignment is extended to a function tr on the set Form of all $\mathcal{L}$-formulas in a straightforward manner. The detailed specification of tr, or rather a slight variant thereof, will be given in a minute.

Before doing so, we recapitulate the language L_2 from [9], Part I, § 1. The set At of all *atomic L_2-formulas* consists of $\top$, all equations $v = v'$, where v, v' are state variables, all predicate expressions $P_p(v)$, where p is a proposition variable and v a state variable, and all membership expressions $v \varepsilon \Upsilon$, where v is as above and Υ a set variable. More complex formulas are obtained from At through (iterated) negation, conjunction, and universal quantification '$\forall v.$' and '$\forall \Upsilon.$' over state and set variables, respectively. Let Fml_2 be the resulting set of formulas. With that, the standard translation announced above is given in the next definition.

Definition 3 (Standard Translation). *Let v be a state variable and Υ a set variable. A mapping $ST_{v,\Upsilon} : \mathsf{Form} \to \mathsf{Fml}_2$ is defined recursively by*

$$\begin{aligned}
ST_{v,\Upsilon}(\top) &:= \top \\
ST_{v,\Upsilon}(p) &:= P_p(v) \\
ST_{v,\Upsilon}(\neg\alpha) &:= \neg ST_{v,\Upsilon}(\alpha) \\
ST_{v,\Upsilon}(\alpha \wedge \beta) &:= ST_{v,\Upsilon}(\alpha) \wedge ST_{v,\Upsilon}(\beta) \\
ST_{v,\Upsilon}(\mathsf{K}\alpha) &:= \forall v'.\, (v' \varepsilon \Upsilon \to ST_{v',\Upsilon}(\alpha)) \\
ST_{v,\Upsilon}(\Box\alpha) &:= \forall \Upsilon'.\, (v \varepsilon \Upsilon' \wedge \forall v.\, (v \varepsilon \Upsilon' \to v \varepsilon \Upsilon) \to ST_{v,\Upsilon'}(\alpha))\,,
\end{aligned}$$

where $p \in \mathsf{Prop}$, $\alpha, \beta \in \mathsf{Form}$, and v', Υ' are variables that have not been used so far in applying ST.

Note that we have, for the last clause of Definition 3, taken advantage of the fact that the subset relation '$\subseteq$' is L_2-definable.

In the following lemma, the intimate correlation between modal and first-order satisfaction is delineated. For that, note that any subset space $\mathcal{M} = (X, \mathcal{O}, V)$ induces an L_2-structure $\mathcal{M}' = \big(X, \mathcal{O}, (V_{P_p})_{p \in \mathsf{Prop}}\big)$ in an obvious way, namely by interpreting P_p with $V(p)$:

$$V_{P_p} = V(p),$$

for every $p \in \mathsf{Prop}$. Therefore, we may identify $\mathcal{M}$ and $\mathcal{M}'$, and we shall do so after the lemma.

Lemma 1. *Let $\mathcal{M} = (X, \mathcal{O}, V)$ be a subset space based on $\mathcal{S} = (X, \mathcal{O})$, and let $\alpha \in \mathsf{Form}$ be a formula. Then we have, for all neighborhood situations $x, U \in \mathcal{N}_\mathcal{S}$ and all look-up tables*[1] *$\mathfrak{b}$ evaluating v to x and Υ to U,*

$$x, U \models_\mathcal{M} \alpha \iff \mathcal{M}' \models ST_{v,\Upsilon}(\alpha)\,(\mathfrak{b})\,.$$

[1] That is, sort-respecting functions mapping variables to values.

The proof of the lemma can easily be carried out by induction on α (omitted here).

The language L_t mentioned in the introduction is a sublanguage of L_2. The only difference comes from quantification over set variables, which is now restricted in the way described below. Call an L_2-formula ϕ *positive in a set variable* Υ if and only if every free occurrence of Υ in ϕ is within the scope of an even number of negation signs. Then,

– if v is a state variable and ϕ is positive in Υ, then $\forall \Upsilon.(v \varepsilon \Upsilon \rightarrow \phi)$ is an L_t-formula.

The importance of L_t lies in the fact that it behaves like a first-order language even when formulas are interpreted in topological spaces; see [9] for more details.

Inspecting Definition 3 for some special cases shows that $ST_{v,\Upsilon}(\alpha)$ may be an L_t-formula, but not in any case. For example, the standard translation of a $\Box\mathsf{K}$-prefixed formula violates the above requirement for quantification. Thus, we notice at this point that there appears to be no obvious syntactic restriction of the formulas in the image of ST.

3 Saturation

Saturation is a technique that is useful for proving the characterization result of Theorem 3 below. Thus, this notion is adapted to the framework of subset spaces in this section.

We first introduce L_2-saturated models. We follow the usual manner of speaking below. Let $\mathcal{M} = (X, \mathcal{O}, V)$ be a subset space (considered as an L_2-structure; see the statement immediately before Lemma 1). Let $A \subseteq X$ and $\mathcal{Q} \subseteq \mathcal{O}$ be any subsets of X and $\mathcal{O}$, respectively, and let $L_2[A, \mathcal{Q}]$ be the language obtained by extending L_2 with new constants $\underline{a}$ for all elements $a \in A$ and $\underline{U}$ for all members $U \in \mathcal{Q}$. The correspondingly expanded structure derived from $\mathcal{M}$ is denoted by $\mathcal{M}_{A,\mathcal{Q}}$. Finally, a set $\Phi(\chi) \subseteq \mathsf{Fml}_2$ of L_2-formulas in which at most the (state or set) variable χ occurs free, is called a *type*. Then, $\mathcal{M}$ is called *ω-saturated* iff, for every pair $(A, \mathcal{Q})$ of *finite* sets $A \subseteq X$ and $\mathcal{Q} \subseteq \mathcal{O}$, and every type $\Phi(\chi)$ that is consistent with the first-order theory of $\mathcal{M}_{A,\mathcal{Q}}$, there exists a *realization* of $\Phi(\chi)$ *in* $\mathcal{M}_{A,\mathcal{Q}}$, i.e., an element $x \in X$ or a subset $U \in \mathcal{O}$ (depending on the sort of χ) such that $\mathcal{M}_{A,\mathcal{Q}} \models \phi(\mathfrak{b})$ for all $\phi \in \Phi$ and all look-up tables $\mathfrak{b}$ mapping χ to x respectively U.

Do ω-saturated L_2-models really exist? – Well, it has already been indicated above that L_2 is a first-order rather than a second-order language so that we can use standard model-theoretic results for answering this question affirmatively. This is to be made a little more precise in the following. For a start, note that a subset space $\mathcal{M} = (X, \mathcal{O}, V)$ is equivalent to a two-sorted first-order structure $\mathcal{M}' = ((X, \mathcal{O}), V_\varepsilon, V)$. Here, $(X, \mathcal{O})$ is the pair of carrier sets of the respective sorts, and V_ε indicates the canonical interpretation of the relation symbol 'ε'; moreover, 'equivalent' means that for every formula $\phi \in \mathsf{Fml}_2$ having, for some

$n \in \mathbb{N}$, its free (state or set) variables among those from the set $\{\chi_1, \ldots, \chi_n\}$, and for all look-up tables $\mathfrak{b}$, it is true that

$$\mathcal{M} \models \phi[\chi_1, \ldots, \chi_n]\,(\mathfrak{b}) \iff \mathcal{M}' \models \phi[\chi_1, \ldots, \chi_n]\,(\mathfrak{b})\,.$$

And vice versa, a two-sorted structure $\mathcal{M} = ((X, Y), V_\varepsilon, V)$ in which V_ε interprets 'ε' somehow, gives rise to an equivalent subset space $\mathcal{M}' = (X, \mathcal{O}, V)$ by letting $\mathcal{O} := \{U' \mid U \in Y\}$, where, given a state variable v and a set variable Υ,

$$U' := \{\mathfrak{b}(v) \in X \mid \mathfrak{b} \text{ a look-up table mapping } \Upsilon \text{ to } U \text{ and } \mathcal{M} \models v\,\varepsilon\,\Upsilon\,(\mathfrak{b})\}.$$

Thus far, we have reduced the L_2-logic of subset spaces to a two-sorted first-order logic L' based on the same language. In particular, the above existence question is thereby shifted to L'.

The way many-sorted logics like L' correspond to usual (i.e., one-sorted) ones is well-known; see, e.g., [11], § 2.1. When translating formulas involving several sorts, quantifications must be relativized and the non-emptiness of the respective carrier sets must be expressed, among other things. And that was about it in the case under consideration, due to the absence of function symbols. Now, in the standard case of first-order logic, an ω-saturated model M^* is obtained from a given model M in such a way that M is *elementarily embeddable* into M^*. This means that M is isomorphic to a substructure of M^* via an injective homomorphism ι, and, after identifying M and $\iota[M]$, the expanded structures M_X and M^*_X, where X is the carrier set of M, satisfy the same first-order sentences; see [6], Chap. 6.1, and cf. [4], Chap. 2.6. One usually writes $\iota : M \preccurlyeq M^*$ in this case. As an immediate consequence we obtain, in particular, that ω-saturated L'-models actually exist. Hence ω-saturated L_2-models exist as well. For a given subset space $\mathcal{M} = (X, \mathcal{O}, V)$, the case $\iota : \mathcal{M} \preccurlyeq \mathcal{M}^* = (X^*, \mathcal{O}^*, V^*)$ with $\mathcal{M}^*$ ω-saturated means, in particular, that $\mathcal{O}^*$ contains the set $\{U^* \mid U \in \mathcal{O}\}$, where each U^* is obtained from the original U via the 'detour' described above in this passage, thus satisfies $\iota[U] = U^* \cap \iota[X]$. Regarding the semantics of subset spaces, the neighborhood situation x^*, U^* is the right image of a given situation x, U under the elementary embedding therefore, where $x^* = \iota(x)$. This will be used later on.

We now turn to *bimodal saturation*. This key concept is introduced in the subsequent definition.

Definition 4 (Bimodal Saturation).

1. *Let $\Sigma \subseteq$ Form be a set of $\mathcal{L}$-formulas, $\mathcal{M} = (X, \mathcal{O}, V)$ a subset space and x, U a neighborhood situation of the underlying frame.*
   (a) *Σ is called* satisfiable in *U iff there exists an element $y \in U$ such that $y, U \models_{\mathcal{M}} \alpha$ for all $\alpha \in \Sigma$. And Σ is called* finitely satisfiable in *U iff every finite subset of Σ is satisfiable in U.*
   (b) *Σ is called* satisfiable in the set of subsets of *U iff there exists an $U' \in \mathcal{O}$ such that $x \in U' \subseteq U$ and $x, U' \models_{\mathcal{M}} \alpha$ for all $\alpha \in \Sigma$. And Σ is called* finitely satisfiable in the set of subsets of *U iff every finite subset of Σ is satisfiable in the set of subsets of U.*

2. *A subset space* $\mathcal{M} = (X, \mathcal{O}, V)$ *is called* bm-saturated *(verbalized* bimodally saturated*), iff the following two conditions are satisfied for all subsets* $\Sigma \subseteq \mathsf{Form}$ *and all neighborhood situations* x, U *of* $(X, \mathcal{O})$.
   (a) *If* Σ *is finitely satisfiable in* U*, then* Σ *is satisfiable in* U.
   (b) *If* Σ *is finitely satisfiable in the set of subsets of* U*, then* Σ *is satisfiable in the set of subsets of* U.

The first important result of this paper states that ω-saturated subset spaces are *bm*-saturated.

Theorem 1. *Let* $\mathcal{M} = (X, \mathcal{O}, V)$ *be an* ω*-saturated subset space. Then,* $\mathcal{M}$ *is bm-saturated.*

Proof. Let $\Sigma \subseteq \mathsf{Form}$, and let x, U be a neighborhood situation of $(X, \mathcal{O})$. We first establish the requirement from item (a) of Definition 4.2. Let Σ be finitely satisfiable in U. Fix a state variable v and a set variable Υ, and define

$$\Sigma' := \{v \,\varepsilon\, \underline{U}\} \cup \{ST_{v,\Upsilon}(\alpha)[\Upsilon \mapsto \underline{U}] \mid \alpha \in \Sigma\},$$

where $ST_{v,\Upsilon}(\alpha)[\Upsilon \mapsto \underline{U}]$ is the L_2-formula resulting from substituting each free occurrence of Υ in $ST_{v,\Upsilon}(\alpha)$ with $\underline{U}$. Since Σ is finitely satisfiable in U, the type Σ' is consistent with the first-order theory of the expanded structure $\mathcal{M}_U$;[2] see Lemma 1. It follows that Σ' is realized in $\mathcal{M}_U$, thanks to the ω-saturatedness of $\mathcal{M}$. In other words, there exists some $y \in X$ such that, for all look-up tables $\mathfrak{b}$ mapping v to y,

$$\mathcal{M}_U \models v \,\varepsilon\, \underline{U}\,(\mathfrak{b}) \text{ and } \mathcal{M}_U \models ST_{v,\Upsilon}(\alpha)[\Upsilon \mapsto \underline{U}]\,(\mathfrak{b}) \text{ for all } \alpha \in \Sigma.$$

Thus, $y \in U$ and $y, U \models_{\mathcal{M}} \alpha$ for all $\alpha \in \Sigma$ (see Lemma 1 again). Consequently, Σ is satisfiable in U.

Secondly, we approach the condition in 4.2 (b). Let Σ be finitely satisfiable in the set of subsets of U. Fix again variables v and Υ as above. The type to be considered in this case is a bit more complicated. We let

$$\Sigma'' := \{\underline{x} \,\varepsilon\, \Upsilon\,,\, \forall v.\,(v \,\varepsilon\, \Upsilon \to v \,\varepsilon\, \underline{U})\} \cup \{ST_{v,\Upsilon}(\alpha)[v \mapsto \underline{x}] \mid \alpha \in \Sigma\}.$$

One can see as above that Σ'' is consistent with the first-order theory of the expanded structure $\mathcal{M}_x$. Hence the ω-saturatedness of $\mathcal{M}$ implies that there is a realization of Σ'' in $\mathcal{M}_x$, say $U' \in \mathcal{O}$. Since

$$\mathcal{M}_x \models \underline{x} \,\varepsilon\, \Upsilon\,(\mathfrak{b}) \text{ and } \mathcal{M}_x \models \forall v.\,(v \,\varepsilon\, \Upsilon \to v \,\varepsilon\, \underline{U})\,(\mathfrak{b})$$

for all look-up tables $\mathfrak{b}$ evaluating Υ to U', we obtain $x \in U' \subseteq U$. And from

$$\mathcal{M}_x \models ST_{v,\Upsilon}(\alpha)[v \mapsto \underline{x}]\,(\mathfrak{b}) \text{ for all such } \mathfrak{b}$$

we conclude that $x, U' \models_{\mathcal{M}} \alpha$ for all $\alpha \in \Sigma$. This proves that Σ is satisfiable in the set of subsets of U. Added together, it follows that $\mathcal{M}$ is *bm*-saturated.

Theorem 1 will be utilized when providing a necessary and sufficient condition for modal equivalence of subset spaces in the next section.

[2] We write $\mathcal{M}_U$ instead of $\mathcal{M}_{\emptyset,\{U\}}$; similar abbreviations are used below, too.

4 Bisimulations

It is well-known that *bisimulation* is the right concept corresponding to the idea of *invariance* of modal formulas with respect to their most common semantics; cf. [4], Ch. 2.2. A respective notion regarding their topological semantics is given in [1], Definition 2.1. And we know bisimulations also for subset spaces since the paper [3].[3] Clearly, the latter are relations between neighborhood situations, as these are the basic semantic entities of $\mathcal{L}$. For convenience of the reader, we shall repeat the definition of subset space bisimulations and state a related lemma. Then, after introducing some notation, we get on to the main result of this section.

Definition 5 (Subset Space Bisimulation). *For $i = 1, 2$, let $\mathcal{S}_i = (X_i, \mathcal{O}_i)$ be subset frames and $\mathcal{M}_i = (X_i, \mathcal{O}_i, V_i)$ subset spaces based on $\mathcal{S}_i$. Moreover, let $R \subseteq \mathcal{N}_{\mathcal{S}_1} \times \mathcal{N}_{\mathcal{S}_2}$ be a non-empty binary relation. Then R is called a* subset space bisimulation (between $\mathcal{M}_1$ and $\mathcal{M}_2$) *iff the following five conditions are satisfied whenever $(x_1, U_1)\, R\, (x_2, U_2)$ is valid for any $(x_1, U_1) \in \mathcal{N}_{\mathcal{S}_1}$ and $(x_2, U_2) \in \mathcal{N}_{\mathcal{S}_2}$:*

1. *For all $p \in \mathsf{Prop}$, $\big(x_1 \in V_1(p) \iff x_2 \in V_2(p)\big)$.*
2. *For all $x_1' \in U_1$ there exists $x_2' \in U_2$ such that $(x_1', U_1)\, R\, (x_2', U_2)$.*
3. *For all $x_2' \in U_2$ there exists $x_1' \in U_1$ such that $(x_1', U_1)\, R\, (x_2', U_2)$.*
4. *If $x_1 \in U_1'$ for any $U_1' \in \mathcal{O}_1$ such that $U_1' \subseteq U_1$, then there exists $U_2' \in \mathcal{O}_2$ satisfying $x_2 \in U_2' \subseteq U_2$ and $(x_1, U_1')\, R\, (x_2, U_2')$.*
5. *If $x_2 \in U_2'$ for any $U_2' \in \mathcal{O}_2$ such that $U_2' \subseteq U_2$, then there exists $U_1' \in \mathcal{O}_1$ satisfying $x_1 \in U_1' \subseteq U_1$ and $(x_1, U_1')\, R\, (x_2, U_2')$.*

We obviously have two so-called *forth conditions* as well as two *back conditions* here, given by items $2, 4$ and $3, 5$ of Definition 5, respectively. This is due to the presence of two modalities, K and $\Box$.

The next lemma shows that subset space bisimulations really entail the invariance of $\mathcal{L}$-formulas.

Lemma 2 (Subset Space Invariance of Formulas). *Let $\mathcal{M}_1 = (X_1, \mathcal{O}_1, V_1)$ and $\mathcal{M}_2 = (X_2, \mathcal{O}_2, V_2)$ be subset spaces based on $\mathcal{S}_1 = (X_1, \mathcal{O}_1)$ and $\mathcal{S}_2 = (X_2, \mathcal{O}_2)$, respectively, and let $R \subseteq \mathcal{N}_{\mathcal{S}_1} \times \mathcal{N}_{\mathcal{S}_2}$ be a subset space bisimulation. Furthermore, assume that $(x_1, U_1) \in \mathcal{N}_{\mathcal{S}_1}$ and $(x_2, U_2) \in \mathcal{N}_{\mathcal{S}_2}$ satisfy $(x_1, U_1)\, R\, (x_2, U_2)$. Then, for all formulas $\alpha \in \mathsf{Form}$, we have that $x_1, U_1 \models_{\mathcal{M}_1} \alpha$ iff $x_2, U_2 \models_{\mathcal{M}_2} \alpha$.*

[3] The variant of this notion considered in Sect. 2.3 of the paper [7] concerns a class of bimodal *Kripke models* associated with subset spaces in a rather obvious way. In case of these so-called *cross axiom models,* the canonical comparison language contains two binary relation symbols belonging to the accessibility relations of the two modalities, K and $\Box$, and is different from L_2 thus. Moreover, not every cross axiom model is induced by a subset space. Therefore, we can admittedly relate the respective bisimulation classes of models, but the classifications by bisimulation in the sense of Theorem 3 below are incomparable (or, not obviously comparable at least).

Proof. The proof is done by an easy induction on the structure of α.

Let $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{M}_1$, $\mathcal{M}_2$, and (x_1, U_1), (x_2, U_2), be as in the previous lemma. The notations stipulated below follow analogous ones from the contexts referred to at the beginning of this section. A triple like $(\mathcal{M}_1, x_1, U_1)$ is called a *situated subset space*. Two situated subset spaces $(\mathcal{M}_1, x_1, U_1)$ and $(\mathcal{M}_2, x_2, U_2)$ are called *modally equivalent* iff, for all $\alpha \in \mathsf{Form}$, it is true that

$$x_1, U_1 \models_{\mathcal{M}_1} \alpha \iff x_2, U_2 \models_{\mathcal{M}_2} \alpha;$$

in this case we write $(\mathcal{M}_1, x_1, U_1) \leftrightsquigarrow (\mathcal{M}_2, x_2, U_2)$. If there exists a subset space bisimulation R between $\mathcal{M}_1$ and $\mathcal{M}_2$ such that $(x_1, U_1)\, R\, (x_2, U_2)$, then we write $(\mathcal{M}_1, x_1, U_1) \underline{\leftrightarrow} (\mathcal{M}_2, x_2, U_2)$. With that, Lemma 2 can be reformulated as follows.

Lemma 3. *Let $(\mathcal{M}_1, x_1, U_1)$ and $(\mathcal{M}_2, x_2, U_2)$ be situated subset spaces. Then, $(\mathcal{M}_1, x_1, U_1) \leftrightsquigarrow (\mathcal{M}_2, x_2, U_2)$ whenever $(\mathcal{M}_1, x_1, U_1) \underline{\leftrightarrow} (\mathcal{M}_2, x_2, U_2)$.*

A class of models satisfying the opposite assertion, i.e., that (modal) equivalence implies bisimilarity, is usually called a *Hennessy-Milner class;* see [4], 2.52.[4] We now prove that the class of all ω-saturated subset spaces is such a Hennessy-Milner class.

Theorem 2. *Let $(\mathcal{M}_1, x_1, U_1)$ and $(\mathcal{M}_2, x_2, U_2)$ be situated subset spaces, and assume that $\mathcal{M}_1$ and $\mathcal{M}_2$ are ω-saturated. Then, $(\mathcal{M}_1, x_1, U_1) \leftrightsquigarrow (\mathcal{M}_2, x_2, U_2)$ implies $(\mathcal{M}_1, x_1, U_1) \underline{\leftrightarrow} (\mathcal{M}_2, x_2, U_2)$.*

Proof. Let $(\mathcal{M}_1, x_1, U_1) \leftrightsquigarrow (\mathcal{M}_2, x_2, U_2)$ be satisfied, and let $\mathcal{M}_1$ and $\mathcal{M}_2$ be based on $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Define a relation $R \subseteq \mathcal{N}_{\mathcal{S}_1} \times \mathcal{N}_{\mathcal{S}_2}$ by $\big((x_1, U_1)\, R\, (x_2, U_2) :\iff (\mathcal{M}_1, x_1, U_1) \leftrightsquigarrow (\mathcal{M}_2, x_2, U_2)\big)$, for all $(x_1, U_1) \in \mathcal{N}_{\mathcal{S}_1}$ and $(x_2, U_2) \in \mathcal{N}_{\mathcal{S}_2}$. Then, $R \neq \emptyset$ according to our assumption. We prove that R is a subset space bisimulation. Only items 2 and 5 of Definition 5 are shown here, because item 1 is clear and the proofs of 3 and 4 are completely analogous. So let $(x_1, U_1)\, R\, (x_2, U_2)$. First, let $x_1' \in U_1$. Define $\Sigma := \{\alpha \in \mathsf{Form} \mid x_1', U_1 \models_{\mathcal{M}_1} \alpha\}$. Consider any finite subset $\Gamma \subseteq \Sigma$. Then, $x_1, U_1 \models_{\mathcal{M}_1} \mathsf{L} \bigwedge \Gamma$. Because of $(x_1, U_1)\, R\, (x_2, U_2)$ we conclude that $x_2, U_2 \models_{\mathcal{M}_2} \mathsf{L} \bigwedge \Gamma$. Consequently, there exists $y_2 \in U_2$ such that $y_2, U_2 \models_{\mathcal{M}_2} \bigwedge \Gamma$. It follows that Σ is finitely satisfiable in U_2. Since $\mathcal{M}_2$ is *bm*-saturated according to Theorem 1, Σ is even satisfiable in U_2. Thus, there exists $x_2' \in U_2$ such that $x_2', U_2 \models_{\mathcal{M}_2} \alpha$ for all $\alpha \in \Sigma$. Since we have $(x, U \not\models_{\mathcal{M}} \beta \iff x, U \models_{\mathcal{M}} \neg\beta)$ in any case, this implies $(x_1', U_1)\, R\, (x_2', U_2)$.

Second, let $U_2' \in \mathcal{O}_2$ be such that $x_2 \in U_2' \subseteq U_2$. We take a similar approach as above, letting $\Sigma := \{\alpha \in \mathsf{Form} \mid x_2, U_2' \models_{\mathcal{M}_2} \alpha\}$ here. With the aid of any finite $\Gamma \subseteq \Sigma$, for which $x_2, U_2 \models_{\mathcal{M}_1} \Diamond \bigwedge \Gamma$ comes about, we conclude that Σ

[4] See also the notes to Chap. 2 of that book, where, beyond the references explaining the origin of that naming, some hints to further characterization theorems of the type claimed here can be found.

is finitely satisfiable in the set of subsets of U_1. Now, the *bm*-saturatedness of $\mathcal{M}_1$ implies that Σ is satisfiable in the set of subsets of U_1. It ensues that there is some $U_1' \in \mathcal{O}_1$ such that $x_1 \in U_1' \subseteq U_1$ and $x_1, U_1' \models_{\mathcal{M}_1} \alpha$ for all $\alpha \in \Sigma$. As above, we get $(x_1, U_1')\, R\, (x_2, U_2')$. This is what we wanted to show.

Making use of Lemma 1 and of both listed in Section 3 immediately before Definition 4, the properties of saturated models and the notations there, we obtain the characterization of modally equivalent subset spaces announced at the end of the previous section as an almost immediate consequence of Lemma 3 and Theorem 2.

Corollary 1. *Let $(\mathcal{M}_1, x_1, U_1)$ and $(\mathcal{M}_2, x_2, U_2)$ be situated subset spaces. Then the following two conditions are equivalent.*

1. $(\mathcal{M}_1, x_1, U_1) \leftrightsquigarrow (\mathcal{M}_2, x_2, U_2)$.
2. *There exist ω-saturated subset spaces $\mathcal{M}_1^*$ and $\mathcal{M}_2^*$ and elementary embeddings $\iota : \mathcal{M}_1 \preccurlyeq \mathcal{M}_1^*$ and $\kappa : \mathcal{M}_2 \preccurlyeq \mathcal{M}_2^*$ such that $x_1, U_1 \overset{\iota}{\mapsto} x_1^*, U_1^*$, $x_2, U_2 \overset{\kappa}{\mapsto} x_2^*, U_2^*$, and $(\mathcal{M}_1^*, x_1^*, U_1^*) \underline{\leftrightarrow} (\mathcal{M}_2^*, x_2^*, U_2^*)$.*

5 The Characterization Theorem

We have nearly finished the preparations for our main result. Only two further lemmata and one additional definition are needed. In the following, the proceeding is similar to that in [4], Sect. 2.6.

Lemma 4. *Let v be a state variable and Υ a set variable, and let $\phi = \phi(v, \Upsilon) \in \mathsf{Fml}_2$ be a formula having at most v and Υ as free variables. Consider the set*

$$modCons(\phi) := \{ST_{v,\Upsilon}(\alpha) \mid \alpha \in \mathsf{Form} \text{ and } \phi(v, \Upsilon) \models ST_{v,\Upsilon}(\alpha)\}$$

of all 'modal consequences' of ϕ. If ϕ is a logical consequence of $modCons(\phi)$, then there exists a modal formula $\beta \in \mathsf{Form}$ such that $\phi(v, \Upsilon) \equiv ST_{v,\Upsilon}(\beta)$.

Proof. Assume that $modCons(\phi) \models \phi(v, \Upsilon)$. Due to the compactness of the logic of subset spaces (see the discussion preceding Definition 4, and cf. [11], § 2.2), there is a finite subset $\Psi \subseteq modCons(\phi)$ satisfying $\Psi \models \phi(v, \Upsilon)$. Thus, we get $\bigwedge \Psi \models \phi(v, \Upsilon)$. On the other hand, we clearly have $\phi(v, \Upsilon) \models \bigwedge \Psi$. Consequently, $\phi(v, \Upsilon) \equiv \bigwedge \Psi$. Since each member of Ψ is a standard translation of a formula from Form, the same is true of $\bigwedge \Psi$. Hence the claimed formula β really exists.

Lemma 5. *Let v, Υ, and ϕ, be as in the previous lemma. Moreover, let $\mathcal{M} = (X, \mathcal{O}, V)$ be a subset space, x, U a neighborhood situation of the underlying frame, and $\mathfrak{B}$ the set of all look-up tables $\mathfrak{b}$ mapping v to x and Υ to U. Assume that, for all $\varphi \in modCons(\phi)$ and all $\mathfrak{b} \in \mathfrak{B}$, $\mathcal{M} \models \varphi(\mathfrak{b})$. Then, joining*

$$\Xi := \{ST_{v,\Upsilon}(\alpha) \mid \alpha \in \mathsf{Form}, \mathcal{M} \models ST_{v,\Upsilon}(\alpha)(\mathfrak{b}) \text{ for all } \mathfrak{b} \in \mathfrak{B}\}$$

with $\{\phi(v, \Upsilon)\}$ results in a consistent set of L_2-formulas.

Proof. Suppose towards a contradiction that this is not the case, i.e., $\Xi \cup \{\phi(v,\Upsilon)\}$ is inconsistent. Then, there exists even a finite subset $\Theta \subseteq \Xi$ such that $\Theta \cup \{\phi(v,\Upsilon)\}$ is inconsistent. We obtain $\models \phi(v,\Upsilon) \rightarrow \neg \bigwedge \Theta$ from that. Consequently, $\phi(v,\Upsilon) \models \neg \bigwedge \Theta$, which implies that $\neg \bigwedge \Theta \in \mathit{modCons}(\phi)$. By assumption, $\mathcal{M} \models \neg \bigwedge \Theta(\mathfrak{b})$ is valid for all $\mathfrak{b} \in \mathfrak{B}$. However, from $\Theta \subseteq \Xi$ we infer that $\mathcal{M} \models \bigwedge \Theta(\mathfrak{b})$ for all $\mathfrak{b} \in \mathfrak{B}$, a contradiction. Thus, the set $\Xi \cup \{\phi(v,\Upsilon)\}$ is consistent.

It is intuitively clear that the invariance of $\mathcal{L}$-formulas under bisimulations (see Lemma 2 or Lemma 3) extends to their standard translations. But we must still put in precise terms what is meant by this notion.

Definition 6 (Invariance of L_2-Formulas under Bisimulations). *Let $\phi = \phi(v,\Upsilon) \in \mathsf{Fml}_2$ be a formula as above. Then ϕ is called* invariant under bisimulations, *iff for all situated subset spaces $(\mathcal{M}_1,x_1,U_1)$ and $(\mathcal{M}_2,x_2,U_2)$ satisfying $(\mathcal{M}_1,x_1,U_1) \leftrightarrows (\mathcal{M}_2,x_2,U_2)$, we have that*

$$\mathcal{M}_1 \models \phi(\mathfrak{b}_1) \iff \mathcal{M}_2 \models \phi(\mathfrak{b}_2)$$

for all look-up tables $\mathfrak{b}_1$, $\mathfrak{b}_2$ mapping v to x_1 and Υ to U_1 and, respectively, v to x_2 and Υ to U_2.

The desired result characterizing the set of all properties that can be specified by $\mathcal{L}$-formulas now reads as follows.

Theorem 3. *A formula $\phi = \phi(v,\Upsilon) \in \mathsf{Fml}_2$ is equivalent to the standard translation of an $\mathcal{L}$-formula if and only if it is invariant under bisimulations.*

Proof. In view of Definition 6, the necessity of the condition is evident.[5] In order to prove its sufficiency, assume that ϕ is invariant under bisimulations. According to Lemma 4, it suffices to prove that ϕ is a logical consequence of $\mathit{modCons}(\phi)$. So let $\mathcal{M} = (X,\mathcal{O},V)$ be a subset space, x,U a neighborhood situation of the underlying frame, $\mathfrak{B}$ the set of all look-up tables $\mathfrak{b}$ mapping v to x and Υ to U, and assume that, for all $\varphi \in \mathit{modCons}(\phi)$ and all $\mathfrak{b} \in \mathfrak{B}$, $\mathcal{M} \models \varphi(\mathfrak{b})$. Lemma 5 guarantees that the set $\Xi \cup \{\phi(v,\Upsilon)\}$ is consistent. Hence there exist a subset space $\mathcal{M}_0 = (X_0,\mathcal{O}_0,V_0)$ and a neighborhood situation x_0,U_0 of $(X_0,\mathcal{O}_0)$ such that, in particular, for all $\varphi \in \mathit{modCons}(\phi)$ and all look-up tables $\mathfrak{b}_0$ mapping v to x_0 and Υ to U_0, $\mathcal{M}_0 \models \varphi(\mathfrak{b}_0)$. We have that $(\mathcal{M},x,U) \leftrightsquigarrow (\mathcal{M}_0,x_0,U_0)$, for on the one hand $x,U \models_{\mathcal{M}} \alpha$ implies $ST_{v,\Upsilon}(\alpha) \in \Xi$, which yields $x_0,U_0 \models_{\mathcal{M}_0} \alpha$, and on the other hand $x,U \not\models_{\mathcal{M}} \alpha$, i.e., $x,U \models_{\mathcal{M}} \neg\alpha$, implies $ST_{v,\Upsilon}(\neg\alpha) \in \Xi$ so that $x_0,U_0 \models_{\mathcal{M}_0} \neg\alpha$, thus $x_0,U_0 \not\models_{\mathcal{M}_0} \alpha$. Taking up Corollary 1 (inclusive of the notations there), we now obtain $(\mathcal{M}^*,x^*,U^*) \leftrightarrows (\mathcal{M}_0^*,x_0^*,U_0^*)$. As we also have $\mathcal{M}_0 \models \phi(\mathfrak{b}_0)$ for all look-up tables $\mathfrak{b}_0$ mapping v to x_0 and Υ to U_0, we get $\mathcal{M}_0^* \models \phi(\mathfrak{b}_0^*)$ for all look-up tables $\mathfrak{b}_0^*$ mapping v to x_0^* and Υ to U_0^*. At this point, the invariance of ϕ under bisimulations comes into play, forcing that $\mathcal{M}^* \models \phi(\mathfrak{b}^*)$ for all look-up tables $\mathfrak{b}^*$ mapping v to x^* and Υ to U^*. It follows that $\mathcal{M} \models \phi(\mathfrak{b})$ for all look-up tables $\mathfrak{b} \in \mathfrak{B}$. In this way, it is established that ϕ is a logical consequence of $\mathit{modCons}(\phi)$. Thus, the theorem is proved.

[5] From now on, we apply Lemma 1 and Lemma 2 without explicit reference.

6 Concluding Remarks

Inspired by the paper [5], we were looking for a van Benthem style characterization of Moss and Parikh's topologic. The actual outcome of this paper is a corresponding result for the logic of subset spaces, having the pseudo-monadic second order language L_2 as its natural comparison. More explicitly, the set of all properties that can be specified by bimodal $\mathcal{L}$-formulas could be classified as the bisimulation-invariant fragment of L_2.

Being a matter of $\mathcal{L}$-*expressivity,* our main result prompts a couple of questions concerning the related field of $\mathcal{L}$-*definability.* Is it possible to characterize definable classes of subset spaces in a similar way as it is the case with ordinary modal logic (see [4], 2.75 and 2.76)? And can a corresponding definability result be proved for *subset frames,* i.e., do we have an appropriate version of the *Goldblatt-Thomason Theorem* (see [4], 3.19) here, too? – These problems will be tackled by future research.

Acknowledgement. I am grateful to the reviewers for their valuable comments and suggestions.

References

1. Aiello, M., van Benthem, J., Bezhanishvili, G.: Reasoning about space: The modal way. Journal of Logic and Computation 13(6), 889–920 (2003)
2. Aiello, M., Pratt-Hartmann, I.E., van Benthem, J.F.A.K.: Handbook of Spatial Logics. Springer (2007)
3. Başkent, C.: Topics in Subset Space Logic. Master's thesis, Institute for Logic, Language and Computation, Universiteit van Amsterdam (July 2007)
4. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, Cambridge (2001)
5. ten Cate, B., Gabelaia, D., Sustretov, D.: Modal languages for topology: Expressivity and definability. Annals of Pure and Applied Logic 159, 146–170 (2009)
6. Chang, C.C., Keisler, H.J.: Model Theory, 3rd edn. Studies in Logic and the Foundations of Mathematics, vol. 73. North-Holland, Amsterdam (1990)
7. Dabrowski, A., Moss, L.S., Parikh, R.: Topological reasoning and the logic of knowledge. Annals of Pure and Applied Logic 78, 73–110 (1996)
8. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge (1995)
9. Flum, J., Ziegler, M.: Topological Model Theory. Lecture Notes in Mathematics, vol. 769. Springer, Berlin (1980)
10. Georgatos, K.: Modal Logics of Topological Spaces. Ph.D. thesis, City University of New York (May 1993)
11. Mal'cev, A.I.: Model Correspondences. In: The Metamathematics of Algebraic Systems. Studies in Logic and the Foundation of Mathematics, ch. 11, vol. 63. North-Holland, Amsterdam (1971)
12. Moss, L.S., Parikh, R.: Topological reasoning and the logic of knowledge. In: Moses, Y. (ed.) Theoretical Aspects of Reasoning about Knowledge (TARK 1992), pp. 95–105. Morgan Kaufmann, Los Altos (1992)

Descriptional Complexity of Operations on Alternating and Boolean Automata*

Galina Jirásková

Mathematical Institute, Slovak Academy of Sciences
Grešákova 6, 040 01 Košice, Slovakia
jiraskov@saske.sk

Abstract. The paper shows that the tight bound for the conversion of alternating finite automata into nondeterministic finite automata with a single initial state is 2^n+1. This solves an open problem stated by Fellah *et al.* (Intern. J. Computer Math. 35, 1990, 117–132). Then we examine the complexity of basic operations on languages represented by boolean and alternating finite automata. We get tight bounds for intersection and union, and for concatenation and reversal of languages represented by boolean automata. In the case of star, and of concatenation and reversal of AFA languages, our upper and lower bounds differ by one.

1 Introduction

Boolean and alternating finite automata [1,2,8,9] are generalizations of nondeterministic finite automata. They recognize regular languages, however, they may be exponentially smaller, in terms of the number of states, than equivalent nondeterministic automata.

Fellah *et al.* [3] studied alternating finite automata (AFAs), that is, boolean automata, in which the initial boolean function is given by a projection. They proved that every AFA of n states can be simulated by a nondeterministic finite automaton *with a single initial state* of at most 2^n+1 states, and left as an open problem the tightness of this upper bound. Our first result provides an answer to this problem by describing an n-state binary AFA whose equivalent NFAs with a single initial state have at least 2^n+1 states.

Then we examine the complexity of basic regular operations on languages represented by boolean and alternating automata. In the case of union and intersection, we get the tight bounds $m+n$ and $m+n+1$ on the boolean and alternating state complexity, respectively. Next we show that the boolean state complexity of concatenation is 2^m+n, and of reversal 2^n. As for the alternating state complexity of concatenation and reversal, our upper and lower bounds differ by one. The same is true for star in both boolean and alternating case.

To get the results, we use known results on the state complexity of operations on regular languages [4,5,6,7,11,14], as well as the fact that if the reversal of a language is accepted by the minimal deterministic automaton of n states, than every boolean automaton for this language has at least $\log n$ states.

* Research supported by VEGA grant 2/0183/11.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 196–204, 2012.

2 Preliminaries

This section gives basic definitions and notations. For all unexplained notions, the reader may refer to [10,12,13].

If Σ is a non-empty finite alphabet, then Σ^* is the set of all strings over Σ, including the empty string ε. A *language* over alphabet Σ is any subset of Σ^*.

A *boolean finite automaton* (BFA) is a quintuple $A = (Q, \Sigma, \delta, g_s, F)$, where Q is a finite non-empty set of states, $Q = \{q_1, \ldots, q_n\}$, Σ is an input alphabet, δ is the transition function that maps $Q \times \Sigma$ into the set $\mathcal{B}_n$ of boolean functions of n boolean variables $q_1, \ldots, q_n$, $g_s \in \mathcal{B}_n$ is the initial boolean function, and $F \subseteq Q$ is the set of final states. For example, let $A_1 = (\{q_1, q_2\}, \{a, b\}, \delta, q_1 \wedge q_2, \{q_2\})$, where transition function δ is given in Table 1.

Table 1. The transition function of boolean automaton A_1

δ	a	b
q_1	$q_1 \vee q_2$	1
q_2	q_2	$q_1 \wedge \overline{q_2}$

The transition function δ is extended to the domain $\mathcal{B}_n \times \Sigma^*$ as follows: For all g in $\mathcal{B}_n$, a in Σ, and w in Σ^*,

$$\begin{aligned}
&\delta(g, \varepsilon) = g;\\
&\text{if } g = g(q_1, \ldots, q_n), \text{ then } \delta(g, a) = g(\delta(q_1, a), \ldots, \delta(q_n, a));\\
&\delta(g, wa) = \delta(\delta(g, w), a).
\end{aligned}$$

Next, let $f = (f_1, \ldots, f_n)$ be the boolean vector with $f_i = 1$ iff $q_i \in F$. The language accepted by BFA A is the set $L(A) = \{w \in \Sigma^* \mid \delta(g_s, w)(f) = 1\}$. In our example we have

$$\begin{aligned}
\delta(g_s, ab) = \delta(q_1 \wedge q_2, ab) = \delta(\delta(q_1 \wedge q_2, a), b) = \delta((q_1 \vee q_2) \wedge q_2, b) =\\
(1 \vee (q_1 \wedge \overline{q_2})) \wedge (q_1 \wedge \overline{q_2}) = q_1 \wedge \overline{q_2}.
\end{aligned}$$

To determine whether $ab \in L(A_1)$, we evaluate $\delta(g_s, ab)$ at vector $f = (0, 1)$. We obtain 0, hence $ab \notin L(A_1)$. On the other hand, we have $abb \in L(A_1)$ since $\delta(g_s, abb) = \delta(q_1 \wedge \overline{q_2}, b) = 1 \wedge (\overline{q_1} \vee q_2) = \overline{q_1} \vee q_2$, which gives 1 at $(0, 1)$.

A boolean finite automaton A is alternating (AFA) if the initial function is given by a projection $g(q_1, \ldots, q_n) = q_i$. It is nondeterministic with multiple initial states (NNFA) if g_s and $\delta(q_k, a)$ are of the form $\bigvee_{i \in I} q_i$. If, moreover, $g_s = q_i$, then automaton A is nondeterministic with a single initial state (NFA). If, moreover, $\delta(q_k, a)$ are of the form q_i, automaton A is deterministic (DFA).

The reverse A^R of NNFA A is obtained from A by swapping the role of the initial and final states, and by reversing all the transitions. The reverse of NNFA A accepts language $L(A)^R = \{w^R \mid w \in L\}$; where w^R is the mirror image of string w defined by $\varepsilon^R = \varepsilon$ and $(wa)^R = aw^R$.

3 Optimal Simulation of AFAs by NFAs

This section provides an answer to an open question stated by Fellah *et al.* [3] whether or not the upper bound $2^n + 1$ on AFA-to-NFA coversion is tight. We start with conversion of boolean automata to NNFAs. Then we show that every NNFA of 2^n states whose reverse is deterministic may be represented by an n-state boolean automaton. This allow us to describe boolean automata by NNFAs in our worst-case example.

Lemma 1. *If a language L is accepted by an n-state boolean automaton, then L is accepted by an NNFA of at most 2^n states.*

Proof. Let $A = (Q, \Sigma, \delta, g_s, F)$ be a boolean automaton with $Q = \{q_1, \ldots, q_n\}$. Like in [3], construct the NNFA $A' = (\{0,1\}^n, \Sigma, \delta', S, \{f\})$, where for every $u = (u_1, \ldots, u_n)$ in $\{0,1\}^n$ and every a in Σ,

$$\begin{aligned} \delta'(u,a) &= \{u' \in \{0,1\}^n \mid \delta(q_i,a)(u') = u_i \text{ for } i = 1, \ldots, n\}, \\ S &= \{b \in \{0,1\}^n \mid g_s(b) = 1\}, \\ f &= (f_1, \ldots, f_n) \in \{0,1\}^n \text{ with } f_i = 1 \text{ iff } q_i \in F. \end{aligned}$$

The proof of $L(A') = L(A)$ is almost the same as in [3, Theorem 4.1]. □

Lemma 2. *Let $A = (Q, \Sigma, \delta, S, F)$ be an NNFA such that $|Q| = 2^n$. Let the reverse of A be deterministic (and complete). Then there exists an n-state boolean automaton A' such that $L(A) = L(A')$. Moreover, if A has 2^{n-1} initial states, then A' may be taken to be alternating.*

Proof. Assume $Q = \{0, 1, \ldots, 2^n - 1\}$. Since A^R is deterministic, NNFA A has exactly one final state, and assume $F = \{k\}$. Moreover, for every symbol a in Σ and every state i in Q, there is *exactly one* state j in Q such that j goes to i by a in NNFA A.

For a state i in Q, let $\mathrm{bin}(i) = (b_1, \ldots, b_n)$ be the binary n-tuple such that $b_1 b_2 \cdots b_n$ is the binary notation of i on n digit with leading zeros if necessary.

Define an n-state boolean automaton $A' = (Q', \Sigma, \delta', g_s, F')$, where $Q' = \{q_1, \ldots, q_n\}$, $F' = \{q_\ell \mid \mathrm{bin}(k)_\ell = 1\}$, and for each i in Q and a in Σ,

$$\begin{gathered} (\delta'(q_1,a), \ldots, \delta'(q_n,a))(\mathrm{bin}(i)) = \mathrm{bin}(j) \text{ where } i \in \delta(j,a), \text{ and} \\ g_s(\mathrm{bin}(i)) = 1 \text{ iff } i \in S. \end{gathered}$$

Then $L(A') = L(A)$. If A has 2^{n-1} initial states, assume $S = \{2^{n-1}, \ldots, 2^n - 1\}$. Now to get an AFA, let $g_s = q_1$, that is $g_s(b_1, \ldots, b_n) = 1$ iff $b_1 = 1$. □

The next lemma shows that there exists an NNFA of 2^n states and 2^{n-1} initial states whose reverse is deterministic, and such that every equivalent NFA requires at least $2^n + 1$ states.

Lemma 3. *Let L be the language accepted by 2^n-state NNFA of Fig. 1. Then every NFA for L has at least $2^n + 1$ states.*

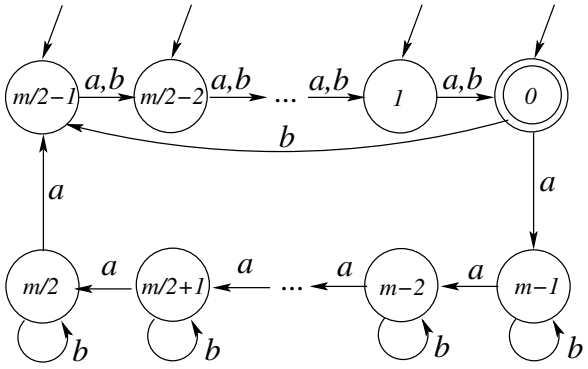


Fig. 1. The NNFA for Lemma 3; $m = 2^n$

Proof. Let $m = 2^n$ and L be the language accepted by the NNFA of Fig. 1. Let N be an NFA (that is, an NNFA with a single initial state) for L. Consider the set of m pairs of strings

$$\mathcal{A} = \{(a^i, a^{m-i}) \mid i = 1, 2, \ldots, m-1\} \cup \{(a^{m-1}b, \varepsilon)\}.$$

For every pair in $\mathcal{A}$, the concatenation of the first part of the pair and its second part results in string a^m or $a^{m-1}b$, both of which are in L. On the other hand, for two distinct pairs in $\mathcal{A}$, the concatenation of the first part of one of the two pairs and the second part of the other pair results in a string in

$$\{a^k \mid m/2 \le k \le m-1 \text{ or } 3m/2 \le k \le 2m-1\} \cup \{a^{m-1}ba^\ell \mid m/2+1 \le \ell \le m-1\}.$$

No such string is in L. It follows that $\mathcal{A}$ is a fooling set for L.

Now fix accepting computations of N on strings $a^i a^{m-i}$ $(1 \le i \le m-1)$ and $a^{m-1}b$, and let p_i $(1 \le i \le m-1)$ and p_m be the states on these computations that are reached after reading a^i resp. $a^{m-1}b$. Since $\mathcal{A}$ is a fooling set for L, states $p_1, \ldots, p_m$ must be pairwise distinct. Now let p_0 be (the sole) initial state of N. Consider the strings $a^{m/2-1}, a, ba^{m/2-1}$ that are in L, so are accepted from state p_0. Then

$$p_0 \notin \{p_n\} \cup \{p_1, \ldots, p_{m/2-1}\}$$

since otherwise one of strings $a^{m-1}b \cdot a^{m/2-1}$ or $a^i \cdot a^{m/2-1}$ with $1 \le i \le m/2-1$ would be accepted by N. However, none of these strings is in L. Next,

$$p_0 \notin \{p_{m/2}, \ldots, p_{m-2}\}$$

since otherwise one of strings $a^j \cdot a$ with $m/2 \le j \le m-2$ would be accepted by N. None of them is in L. Finally, $p_0 \ne p_{m-1}$ since otherwise the string $a^{m-1} \cdot ba^{m/2-1}$, that is not in L, would be accepted by N. Therefore, NFA N must have at least $m+1$ states, and the lemma follows. □

The next result gives the optimal simulation of boolean automata by NFAs. The worst-case language is accepted by an AFA over a two-letter alphabet.

Theorem 1. *Let L be a language accepted by an n-state boolean automaton. Then $2^n + 1$ states are sufficient and necessary in the worst case for nondeterministic finite automata with a single initial state to accept language L. The upper bound is met by an n-state* alternating *automaton over a* binary *alphabet.*

Proof. By Lemma 1, language L is accepted by an 2^n-state NNFA. By adding a new initial state going by the empty string to the initial states of the NNFA, we get an NFA of $2^n + 1$ states for L.

For tightness, consider the language L accepted by the 2^n-state NNFA of Fig. 1. The NNFA has 2^{n-1} initial states, and its reverse is deterministic. By Lemma 2, language L is accepted by an n-state AFA. By Lemma 3, every NFA for L has at least $2^n + 1$ states. This proves the theorem. □

4 Boolean and Alternating State Complexity of Basic Regular Operations

This section examines complexity of basic regular operations on languages represented by boolean and alternating automata.

Recall that the state complexity of a regular language L, $\mathrm{sc}(L)$, is the smallest number of states in any DFA accepting L. Similarly define nondeterministic, alternating, and boolean state complexity of a regular language L, in short $\mathrm{nsc}(L), \mathrm{asc}(L)$, and $\mathrm{bsc}(L)$, respectively, as the smallest number of states in any NFA, AFA, and BFA for L, respectively. The following results are well known.

Lemma 4 ([1,9,3]). *If L is accepted by a boolean automaton of n-states, then L^R is accepted by a DFA of 2^n states. If L is accepted by an AFA of n-states, then L^R accepted by DFA of 2^n states of which 2^{n-1} are final. If $\mathrm{sc}(L^R) = 2^n$ then $\mathrm{bsc}(L) \geq n$. If $\mathrm{sc}(L^R) = 2^n$ and minimal dfa for L^R has more than 2^{n-1} or less than 2^{n-1} final states, then $\mathrm{asc}(L) \geq n + 1$.*

We start with union and intersection, and show that the boolean state complexity of both operations is $m+n$, while their alternating state complexity is $m+n+1$.

Theorem 2 (Union and Intersection on BFAs). *Let K and L be languages over an alphabet Σ with $\mathrm{bsc}(K) = m$ and $\mathrm{bsc}(L) = n$. Then*

1) $\mathrm{bsc}(K \cup L) \leq m + n$,

2) $\mathrm{bsc}(K \cap L) \leq m + n$,

and both bounds are tight if $|\Sigma| \geq 2$.

Proof. Let languages K and L be accepted by BFAs $(Q_A, \Sigma, \delta_A, g_A, F_A)$ and $(Q_B, \Sigma, \delta_B, g_B, F_B)$, respectively. Let $|Q_A| = m$, $|Q_B| = n$, and $Q_A \cap Q_B = \varnothing$. Then the languages $K \cup L$ and $K \cap L$ are accepted by boolean automata $(Q_A \cup Q_B, \Sigma, \delta, g_A \vee g_B, F_A \cup F_B)$ and $(Q_A \cup Q_B, \Sigma, \delta, g_A \wedge g_B, F_A \cup F_B)$, resp., where $\delta(p, a) = \delta_A(p, a)$ if $p \in Q_A$ and $\delta(p, a) = \delta_B(p, a)$ if $p \in Q_B$.

For tightness, consider languages

$$K^R = \{w \in \{a,b\}^* \mid |w|_a \equiv 0 \bmod 2^m\}, \text{ and}$$
$$L^R = \{w \in \{a,b\}^* \mid |w|_b \equiv 0 \bmod 2^n\}.$$

Both languages are accepted by DFAs of 2^m and 2^n states, respectively. Therefore, languages K and L are accepted by boolean automata of m and n states, respectively. Next we have $(K \cup L)^R = K^R \cup L^R$, and it is known [5,11,14] that the minimal DFA for $K^R \cup L^R$ has 2^{m+n} states. It follows that every boolean automaton for $K \cup L$ has at least $m+n$ states. The same argument holds for intersection. □

Theorem 3 (Union and Intersection on AFAs). *Let K and L be languages over an alphabet Σ with* $\mathrm{asc}(K) = m$ *and* $\mathrm{asc}(L) = n$. *Then*
1) $\mathrm{asc}(K \cup L) \leq m+n+1$,
2) $\mathrm{asc}(K \cap L) \leq m+n+1$,
and both bounds are tight if $|\Sigma| \geq 2$.

Proof. The upper bounds are from [3]. For tightness, consider languages $(K')^R$ and $(L')^R$ accepted by DFAs obtained from the DFAs for K^R and L^R in the previous lemma by making states $2^{m-1}, \ldots, 2^m - 1$ in the DFA for K, and states $2^{n-1}, \ldots, 2^n - 1$ in the DFA for L final. Since both DFAs have half of states final, languages K' and L' are accepted by alternating finite automata of m and n states, respectively. To accept language $(K' \cup L')^R$, we still need 2^{m+n} deterministic states, but this time, the number of final states in the minimal DFA for $(K' \cup L')^R$ is more than 2^{m+n-1}. It follows that the minimal AFA for $(K' \cup L')$ has at least $m+n+1$ states. In the case of intersection, the minimal 2^{m+n}-state DFA for $(K' \cap L')^R$ has less then 2^{m+n-1} final states, so $\mathrm{asc}(K' \cap L') \geq m+n+1$. □

Now we consider concatenation. First we show that its boolean state complexity is $2^m + n$. In the case of alternating state complexity, we get an upper bound $2^m + n + 1$, and a lower bound $2^m + n$.

Theorem 4 (Concatenation on BFAs). *Let K and L be languages over an alphabet Σ with* $\mathrm{bsc}(K) = m$ *and* $\mathrm{bsc}(L) = n$. *Then* $\mathrm{bsc}(KL) \leq 2^m + n$, *and the bound is tight if* $|\Sigma| \geq 2$.

Proof. To prove the upper bound, first transform the BFA for K into an NNFA with 2^m states which accepts language K. Then, using idea in [3, Theorem 9.2], we get a boolean automaton of $2^m + n$ states for language KL.

We now prove tightness. It is well known that the tight bound on the state complexity of concatenation is $(m-1)2^n + 2^{n-1}$ and the bound is met by binary DFA languages [11,14]. Now let L^R and K^R be the Maslov's [11] binary witnesses for concatenation with 2^n and 2^m states, respectively. Then $\mathrm{bsc}(K) \leq m$ and $\mathrm{bsc}(L) \leq n$. Moreover, $\mathrm{sc}(KL)^R = \mathrm{sc}(L^R K^R) = (2^n - 1) \cdot 2^{2^m} + 2^{2^m - 1} \geq 2^{n-1} \cdot 2^{2^m} + 2^{n-1} \cdot 2^{2^m - 1} \geq 2^{n-1} 2^{2^m} (1 + 1/2)$. It follows that $\mathrm{bsc}(KL) \geq \lceil \log(2^{n-1} 2^{2^m} (1 + 1/2)) \rceil = 2^m + n$, and the theorem is proved. □

Theorem 5 (Concatenation on AFAs). *Let K and L be languages over an alphabet Σ with* $\mathrm{asc}(K) = m$ *and* $\mathrm{asc}(L) = n$. *Then* $\mathrm{asc}(KL) \leq 2^m + n + 1$. *The bound $2^m + n$ is met if $|\Sigma| \geq 2$.*

Proof. The upper bound is from [3]. For tightness, we need L^R and K^R with half of states final. In such a case, the upper bound for concatenation is $m/2 \cdot 2^n + m/2 \cdot 2^{n-1}$ [14] and is met by binary languages [4]. Now let L^R and K^R be the binary witnesses from [4] with 2^n and 2^m states, respectively, half of which are final in both DFAs. Then $\mathrm{asc}(K) \leq m$ and $\mathrm{asc}(L) \leq n$. The minimal DFA for $(KL)^R = L^R K^R$ has $2^{n-1} \cdot 2^{2^m} + 2^{n-1} \cdot 2^{2^m-1}$, so $\mathrm{asc}(KL) \geq 2^m + n$. This completes the proof. □

The next two results show that the boolean state complexity of reversal is 2^n, while its alternating state complexity is at least 2^n and at most $2^n + 1$.

Theorem 6 (Reversal on BFAs). *Let L be a language over an alphabet Σ with* $\mathrm{bsc}(L) = n$. *Then* $\mathrm{bsc}(L^R) \leq 2^n$, *and the bound is tight if $|\Sigma| \geq 2$.*

Proof. If L is accepted by an n-state boolean automaton, then L^R is accepted by an 2^n-state NNFA. Thus $\mathrm{bsc}(L^R) \leq 2^n$. The upper bound on the state complexity of reversal of regular languages is 2^n, and it is known to be tight in the binary case [7,9]. Let L^R be the binary witness from [7, Theorem 5] with 2^n states. Then $\mathrm{bsc}(L) \leq n$, and $\mathrm{sc}((L^R)^R) = 2^{2^n}$. Therefore, $\mathrm{bsc}(L^R) \geq 2^n$, and our proof is complete. □

Theorem 7 (Reversal on AFAs). *Let L be a language over an alphabet Σ with* $\mathrm{asc}(L) = n$. *Then* $\mathrm{asc}(L^R) \leq 2^n + 1$. *The bound 2^n is met if $|\Sigma| \geq 2$.*

Proof. If L is accepted by an n-state AFA, then L^R is accepted by an $(2^n + 1)$-state NFA. Thus $\mathrm{asc}(L^R) \leq 2^n + 1$. For the lower bound, we need a witness for reversal with half of states final. Such an example is given in [9, Proposition 2] with 2^n states. This proves lower bound and completes the proof. □

The last operation under consideration is star operation The following two theorems show that both boolean and alternating complexity of star are at least 2^n and at most $2^n + 1$.

Theorem 8 (Star on BFAs). *Let L be a language over an alphabet Σ with* $\mathrm{bsc}(L) = n$. *Then* $\mathrm{bsc}(L^*) \leq 2^n + 1$. *The bound 2^n is met if $|\Sigma| \geq 2$.*

Proof. If L is accepted by an n-state boolean automaton, then L^* is accepted by an $(2^n + 1)$-state NFA. Thus $\mathrm{bsc}(L^*) \leq 2^n + 1$. The upper bound on the state complexity of star of regular languages is $2^{n-1} + 2^{n-2}$, and is known to be tight in the binary case [14]. Let L^R be the binary witness from [14] with 2^n states. Then $\mathrm{bsc}(L) \leq n$, and

$$\mathrm{sc}((L^*)^R) = \mathrm{sc}((L^R)^*) = 2^{2^n-1} + 2^{2^n-2} = 2^{2^n-1}(1 + 2^{-1}).$$

Hence $\mathrm{bsc}(L^*) \geq 2^n$, and the theorem follows. □

Theorem 9 (Star on AFAs). *Let L be a language over an alphabet Σ with* $\mathrm{asc}(L) = n$. *Then* $\mathrm{asc}(L^*) \leq 2^n + 1$. *The bound* 2^n *is met if* $|\Sigma| \geq 2$.

Proof. The upper bound follows like in the previous theorem. For the lower bound, we need L^R with half of states final. In such a case, the upper bound for star is $2^{n-1} + 2^{n-1-\ell}$, where ℓ is the number of final states different from the initial state. Such a bound is met for every ℓ $(1 \leq \ell \leq n-1)$ in the binary case [6]. Let L^R be the binary witness from [6, Theorem 5] with 2^n states and $\ell = 2^{n-1}$ final states distinct from the initial state. Then $\mathrm{asc}(L) \leq n$, and

$$\mathrm{sc}((L^*)^R) = \mathrm{sc}((L^R)^*) = 2^{2^n-1} + 2^{2^n-1-n/2} = 2^{2^n-1}(1 + 2^{-n/2}).$$

Thus $\mathrm{asc}(L^*) \geq 2^n$, which proves the theorem. □

5 Conclusions

We proved the tight bound on the number of states of nondeterministic finite automaton with a single initial state that are sufficient and necessary in the worst case to simulate a boolean or alternating finite automaton of n states is $2^n + 1$. This solves an open problem from [3].

Then we examined the boolean and alternating state complexity of basic regular operations. Our results are summarised in the following table. All the worst-case examples are defined over a binary alphabet. The tightness of the upper bounds for star of BFAs and AFAs, as well as for concatenation and reversal of AFAs, remains open.

Table 2. Boolean and alternating state complexity of basic regular operations

	union	intersection	concatenation	reversal	star
BFAs	$m+n$	$m+n$	2^m+n	2^n	$\geq 2^n$ $\leq 2^n+1$
AFAs	$m+n+1$	$m+n+1$	$\geq 2^m+n$ $\leq 2^m+n+1$	$\geq 2^n$ $\leq 2^n+1$	$\geq 2^n$ $\leq 2^n+1$

Acknowledgement. I would like to thank Markus Holzer and Martin Kutrib for reading a preliminary version of the proof of Theorem 1, and for interesting discussions on the topic.

References

1. Brzozowski, J., Leiss, E.: On equations for regular languages, finite automata, and sequential networks. Theoret. Comput. Sci. 10, 19–35 (1980)
2. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. JACM 28, 114–133 (1981)

3. Fellah, A., Jürgensen, H., Yu, S.: Constructions for alternating finite automata. Intern. J. Computer Math. 35, 117–132 (1990)
4. Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation. Internat. J. Found. Comput. Sci. 16, 511–529 (2005)
5. Jirásková, G., Masopust, T.: Complexity in Union-Free Regular Languages. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 255–266. Springer, Heidelberg (2010)
6. Jirásková, G., Masopust, T.: State Complexity of Projected Languages. In: Holzer, M., Kutrib, M., Pighizzini, G. (eds.) DCFS 2011. LNCS, vol. 6808, pp. 198–211. Springer, Heidelberg (2011)
7. Jirásková, G., Šebej, J.: Note on Reversal of Binary Regular Languages. In: Holzer, M., Kutrib, M., Pighizzini, G. (eds.) DCFS 2011. LNCS, vol. 6808, pp. 212–221. Springer, Heidelberg (2011)
8. Kozen, D.: On parallelism in turing machines. In: Proc. 17th FOCS, pp. 89–97 (1976)
9. Leiss, E.: Succinct representation of regular languages by boolean automata. Theoret. Comput. Sci. 13, 323–330 (1981)
10. Leiss, E.: On generalized language equations. Theoret. Comput. Sci. 14, 63–77 (1981)
11. Maslov, A.N.: Estimates of the number of states of finite automata. Soviet Math. Doklady 11, 1373–1375 (1970)
12. Sipser, M.: Introduction to the theory of computation. PWS Publishing Company, Boston (1997)
13. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, ch. 2, vol. I, pp. 41–110. Springer, Heidelberg (1997)
14. Yu, S., Zhuang, Q., Salomaa, K.: The state complexity of some basic operations on regular languages. Theoret. Comput. Sci. 125, 315–328 (1994)

Consistency of Multidimensional Combinatorial Substitutions

Timo Jolivet[1,2] and Jarkko Kari[1]

[1] FUNDIM, Department of Mathematics, University of Turku, Finland
[2] LIAFA, Université Paris 7, France

Abstract. Multidimensional combinatorial substitutions are rules that replace symbols by finite patterns of symbols in $\mathbb{Z}^d$. We focus on the case where the patterns are not necessarily rectangular, which requires a specific description of the way they are glued together in the image by a substitution. Two problems can arise when defining a substitution in such a way: it can fail to be consistent, and the patterns in an image by the substitution might overlap.

We prove that it is undecidable whether a two-dimensional substitution is consistent or overlapping, and we provide practical algorithms to decide these properties in some particular cases.

1 Introduction

One-dimensional substitutions are a classical object of combinatorics on words. An example is the map $\sigma : \{1,2,3\}^* \to \{1,2,3\}^*$ defined by $\sigma(1) = 12$, $\sigma(2) = 13$ and $\sigma(3) = 1$. The image by σ of a word is easy to define, by *concatenation*: $\sigma(uv) = \sigma(u)\sigma(v)$ for all $u, v \in \{1,2,3\}^*$. For example, $\sigma(1321) = 1211312$. See [1] for a detailed survey.

A natural generalization of substitutions to higher dimension is the case where the images of the letters are squares of the same size, as for example in the two-dimensional Thue-Morse substitution defined by $1 \mapsto \begin{smallmatrix}1&2\\2&1\end{smallmatrix}$, $2 \mapsto \begin{smallmatrix}2&1\\1&2\end{smallmatrix}$, which can be iterated naturally, as shown below.

$$1 \mapsto \begin{smallmatrix}1&2\\2&1\end{smallmatrix} \mapsto \begin{smallmatrix}1&2&2&1\\2&1&1&2\\2&1&1&2\\1&2&2&1\end{smallmatrix} \mapsto \begin{smallmatrix}1&2&2&1&2&1&1&2\\2&1&1&2&1&2&2&1\\2&1&1&2&1&2&2&1\\1&2&2&1&2&1&1&2\\1&2&2&1&2&1&1&2\\2&1&1&2&1&2&2&1\\2&1&1&2&1&2&2&1\\1&2&2&1&2&1&1&2\end{smallmatrix}$$

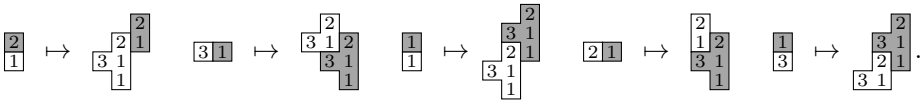
Similar generalizations are possible with rectangular shapes that have compatible edge lengths; see the survey [2].

We are interested in the more general case where the images of the letters have arbitrary shapes (not necessarily rectangular), which are *a priori* not compatible with concatenation. For example, if σ is a substitution defined by:

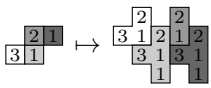
$$\boxed{1} \mapsto \begin{smallmatrix}&\boxed{2}\\\boxed{3}&1\\&\boxed{1}\end{smallmatrix} \qquad \boxed{2} \mapsto \begin{smallmatrix}\boxed{2}\\\boxed{1}\end{smallmatrix} \qquad \boxed{3} \mapsto \begin{smallmatrix}&\boxed{2}\\\boxed{3}&\boxed{1}\end{smallmatrix},$$

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 205–216, 2012.

how can we define the image by σ of patterns that consist of more than one cell? A natural approach is to give explicit *concatenation rules*, such as:



Using these rules, we can compute the image of every pattern that can be "covered" by the two-cell patterns on the left-hand sides of the rules, as shown below.



In this paper we study the two problems that can arise when multidimensional concatenation is defined as above:

1. The resulting substitution is not necessarily *consistent*: depending on the sequence of concatenation rules that are used, a pattern might have two different images.
2. The resulting substitution is not necessarily *non-overlapping*: the images of the cells of a pattern can overlap.

The substitutions which are consistent and non-overlapping correspond to "well defined" substitutions, and we would like to be able to detect them algorithmically. We will prove that consistency and non-overlapping are undecidable properties for two-dimensional combinatorial substitutions (Theorems 3.1 and 3.3), but that these properties are decidable in the case of *domino-complete* substitutions, that is, when the concatenation rules are given for every possible domino (Theorems 4.1 and 4.4). This answers the decidability question raised in [3].

Combinatorial substitutions as defined in this article were introduced in [4] under the name "local rules", to study a particular example in the context of substitutions of unit faces of discrete planes. They were defined in generality in [3], and are related to the substitutions found in [5] defined using the dual graph of a pattern. See [2] for a survey about multidimensional substitutions.

As an application of the methods developed for the decidability results, we provide combinatorial proofs of the consistency and non-overlapping of some particular two-dimensional substitutions, using a slightly more general definition of domino-completeness. Such proofs have been requested in [4,2]. The study of this particular class of examples will be published in [6], which also contains additional proofs and examples.

2 Definitions

Cells and Patterns. Let $\mathcal{A}$ denote a set of symbols. A *d-dimensional cell* is a couple $c = [v, t]$, where $v \in \mathbb{Z}^d$ is the *vector* of c and $t \in \mathcal{A}$ is the *type* of c. A *d-dimensional pattern* is a finite union of d-dimensional cells with distinct

vectors. Translation $P + v$ of a pattern P by $v \in \mathbb{Z}^d$ is defined in the natural way. The *support* of a pattern is $\mathrm{supp}(P) = \{v : [v, t] \in P\}$.

Many of the substitutions we will encounter later use *dominoes*, which are two-dimensional patterns that consists of two cells of vectors v and v' such that $v' - v \in \{(\pm 1, 0), (0, \pm 1))\}$.

Substitutions. A *d-dimensional substitution* σ on alphabet $\mathcal{A}$ is defined by:

- a *base rule*: an application σ_{base} from $\mathcal{A}$ to the set of d-dimensional patterns,
- a finite set of *concatenation rules* $(t, t', u) \mapsto v$, where $t, t' \in \mathcal{A}$ and $u, v \in \mathbb{Z}^d$.

The way to interpret this definition is the following: σ_{base} replaces each cell of a pattern by a pattern, and the concatenation rules describe how to place the images of the cells relatively to each other. The intuitive meaning of "$(t, t', u) \mapsto v$" is: two cells of types t and t' separated by u must be mapped by σ to the two patterns $\sigma_{\mathrm{base}}(t)$ and $\sigma_{\mathrm{base}}(t')$ separated by v. (A precise definition is given below.) From now on we only consider *deterministic* substitutions, which means that if a rule has a left-hand side (t, t', u), then there is no other rule with left-hand side either (t, t', u) or $(t', t, -u)$.

We will need the following notation:

- We extend σ_{base} from $\mathcal{A}$ to the set of cells naturally: $\sigma_{\mathrm{base}}(c) = \sigma_{\mathrm{base}}(t)$ for a cell $c = [v, t]$. (Only the type of c is taken into account by σ_{base}.)
- The set of the *starting patterns* of σ is $\mathcal{C}_\sigma = \{\{[0, t], [u, t']\} : (t, t', u) \mapsto v \text{ is a rule of } \sigma \text{ for some } v\}$. It corresponds to the patterns at the left-hand sides of the concatenation rules of σ. (This set contains only patterns of size two.)
- If $c = [u, t]$ and $c' = [u', t']$ are cells, we denote

$$\sigma_{\mathrm{rule}}(c, c') \;=\; \begin{cases} v & \text{if } (t, t', u' - u) \mapsto v \text{ is a rule of } \sigma \\ -v & \text{if } (t', t, u - u') \mapsto v \text{ is a rule of } \sigma \end{cases},$$

  and σ_{rule} is not defined otherwise.

A *domino substitution* is a two-dimensional substitution such that for every rule $(t, t', u) \mapsto v$, we have $u \in \{\pm(1, 0), \pm(0, 1)\}$. A *domino-to-domino* substitution is a domino substitution such that for every rule $(t, t', u) \mapsto v$, we have $v \in \{\pm(1, 0), \pm(0, 1)\}$, and the patterns $\sigma_{\mathrm{base}}(t)$ and $\sigma_{\mathrm{base}}(t')$ both consist of a single cell of vector $(0, 0)$.

Example 2.1. The combinatorial substitution given in the introduction is formally defined as follows: it is the two-dimensional substitution defined on the alphabet $\{1, 2, 3\}$ with the following base rule (on the left) and concatenation rules (on the right).

$$\begin{array}{ll} & (1, 2, (0, 1)) \mapsto (1, 2) \\ 1 \mapsto \{[(0, 0), 1], [(0, 1), 1], [(0, 2), 2], [(-1, 1), 3]\} & (3, 1, (1, 0)) \mapsto (2, -2) \\ 2 \mapsto \{[(0, 0), 1], [(0, 1), 2]\} & (1, 1, (0, 1)) \mapsto (1, 2) \\ 3 \mapsto \{[(0, 0), 3], [(1, 0), 1], [(1, 1), 2]\} & (2, 1, (1, 0)) \mapsto (1, -2) \\ & (3, 1, (0, 1)) \mapsto (2, 1). \end{array}$$

Paths, Covers, Image Vectors. Let $\mathcal{C}$ be a finite set of patterns that consist of two cells. A *$\mathcal{C}$-path from cell c_1 to cell c_n* is a finite sequence of cells $\gamma = (c_1, \dots, c_n)$ such that $\{c_i, c_{i+1}\}$ is a translated copy of an element of $\mathcal{C}$ for all $1 \leqslant i \leqslant n-1$, and such that $c_i = c_j$ if c_i and c_j have the same vector. (Paths are hence allowed to self-overlap, but the overlapping cells must agree.) If $c_1 = c_n$, then γ is called a *$\mathcal{C}$-loop*. A path (or a loop) is *simple* if it does not self-intersect. A *domino path* is a $\mathcal{C}$-path where all the patterns of $\mathcal{C}$ are dominoes. Most of the paths we will consider in this paper will be domino paths (associated with some domino substitutions).

A $\mathcal{C}$-path *of a pattern P* is a $\mathcal{C}$-path whose cells are all contained in P. We say that P is *$\mathcal{C}$-covered* if for every $c, c' \in P$, there exists a $\mathcal{C}$-path of P from c to c'.

Let σ be a substitution and $\gamma = (c_1, \dots, c_n)$ be a $\mathcal{C}_\sigma$-path. We denote by $\omega_\sigma(\gamma)$ the *image vector of* γ defined by

$$\omega_\sigma(\gamma) \;=\; \sum_{i=1}^{n-1} \sigma_{\text{rule}}(c_i, c_{i+1}).$$

Consistency and Non-Overlapping. Let σ be a substitution and P be a $\mathcal{C}_\sigma$-covered pattern. We say that σ is:

- *consistent on P* if for every cells $c, c' \in P$ and for every $\mathcal{C}_\sigma$-paths γ, γ' of P from c to c', we have $\omega_\sigma(\gamma) = \omega_\sigma(\gamma')$, *i.e.*, if the placement of the images does not depend on the path used.
- *non-overlapping on P* if for every cells $c, c' \in P$ such that $c \neq c'$ and for every $\mathcal{C}_\sigma$-path γ of P from c to c', we have $\text{supp}(\sigma_{\text{base}}(c)) \cap (\text{supp}(\omega_\sigma(\gamma) + \sigma_{\text{base}}(c'))) = \varnothing$, *i.e.*, if two distinct cells have non-overlapping images.

If σ is consistent on every $\mathcal{C}_\sigma$-covered pattern, then σ is said to be *consistent*. (The same goes for non-overlapping.) Examples of inconsistent and overlapping substitutions will be given in Examples 2.4 and 2.5.

Proposition 2.2. *Let σ be a substitution and P be a pattern. The following statements are equivalent.*

1. *σ is consistent on P.*
2. *For every $\mathcal{C}_\sigma$-loop γ of P, we have $\omega_\sigma(\gamma) = 0$.*
3. *For every simple $\mathcal{C}_\sigma$-loop γ of P, we have $\omega_\sigma(\gamma) = 0$.*

The proof of Proposition 2.2 is a direct application of the definitions.

Image by a substitution. Let σ be a non-overlapping substitution. Let P be a $\mathcal{C}_\sigma$-covered pattern and c_0 be a cell of P. An *image of P by σ computed from* c_0 is a pattern

$$\bigcup_{c \in P} (\sigma_{\text{base}}(c) \;+\; \omega_\sigma(\gamma_c)),$$

where for each $c \in P$, γ_c is a $\mathcal{C}_\sigma$-path from c_0 to c. This union of patterns is indeed a pattern because the cells have distinct positions (σ is non-overlapping).

If σ is consistent, this pattern is uniquely defined because it does not depend on the choice of the paths γ_c, by consistency of σ. In this case, the image of P by σ computed from c_0 is denoted by $\sigma(P, c_0)$, but because c_0 only affects by a translation we will use the simpler notation $\sigma(P)$ when the translation is irrelevant.

To explicitly compute the image of an $\mathcal{C}_\sigma$-covered pattern P, we start by constructing a tree whose vertices are the cells of P, where two cells are connected if they both belong to a same pattern of $\mathcal{C}_\sigma$. (Such a connected tree exists because P is $\mathcal{C}_\sigma$-covered.) We then choose a "root cell" (c_0 in the definition above), and we construct the image of P incrementally, starting from c_0 and following the edges in the tree. This is shown in Example 2.3 below.

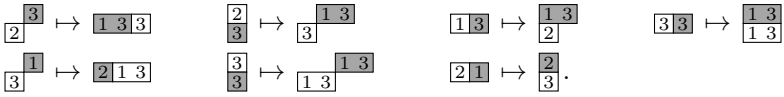
Example 2.3. Let σ be the two-dimensional substitution defined on the alphabet $\{1, 2, 3\}$ with the base rule

$$1 \mapsto \{[(0,0),2]\} \qquad 2 \mapsto \{[(0,0),3]\} \qquad 3 \mapsto \{[(0,0),1],[(1,0),3]\}$$

and the concatenation rules

$$\begin{array}{llll} (2,3,\binom{1}{1}) \mapsto \binom{-2}{0} & (3,2,\binom{0}{1}) \mapsto \binom{-1}{-1} & (1,3,\binom{1}{0}) \mapsto \binom{0}{1} & (3,3,\binom{1}{0}) \mapsto \binom{0}{1} \\ (3,1,\binom{1}{1}) \mapsto \binom{-1}{0} & (3,3,\binom{0}{1}) \mapsto \binom{-2}{-1} & (2,1,\binom{1}{0}) \mapsto \binom{0}{1} & \end{array}$$

which can also be represented as follows

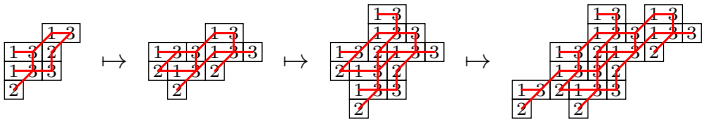


To compute the image of $\begin{array}{|c|c|c|}\hline 1&3&3\\\hline 2&1&3\\\hline\end{array}$ by σ, we can use the $\mathcal{C}_\sigma$-covering $\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 2&1&3\\\hline\end{array}$:

$$\begin{aligned} \sigma\left(\begin{array}{|c|c|c|}\hline 1&3&3\\\hline 2&1&3\\\hline\end{array}\right) &= \{[0,0],3\} \\ &\cup \{[(0,0),1],[(1,0),3]\} + (-2,0) \\ &\cup \{[(0,0),2]\} + (-2,0) + (0,-1) \\ &\cup \{[(0,0),1],[(1,0),3]\} + (-2,0) + (0,1) \\ &\cup \{[(0,0),1],[(1,0),3]\} + (-2,0) + (0,1) + (2,1) \\ &\cup \{[(0,0),2]\} + (-2,0) + (0,1) + (2,1) + (0,-1) \\ &= \{[(0,0),3],[(-2,0),1],[(-1,0),3],[(-2,-1),2], \\ &\quad [(-2,1),1],[(-1,1),3],[(0,1),2],[(0,2),1],[(1,2),3]\} \\ &= \begin{array}{|c|c|c|}\hline &1&3\\\hline 1&3&2\\\hline 1&3&3\\\hline 2\\\cline{1-1}\end{array}\ . \end{aligned}$$

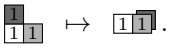
We chose to place the image of the cell $\boxed{2}$ first, at position $(0,0)$. This cell and its image are shown in gray. In this case it is possible to iterate σ on its images;

three iterations are shown below. (The $\mathcal{C}_\sigma$-covering is drawn on each pattern.)



Example 2.4. The substitution defined by the base rule $1 \mapsto \{[(0,0),1\}$, $2 \mapsto \{[(0,0),2\}$ and the concatenation rules $\boxed{1}\boxed{1} \mapsto \boxed{1}\boxed{1}$, $\begin{smallmatrix}1\\2\end{smallmatrix} \mapsto \begin{smallmatrix}1\\2\end{smallmatrix}$, $\boxed{2}\boxed{2} \mapsto \begin{smallmatrix}2\\2\end{smallmatrix}$ is not consistent: the $\mathcal{C}_\sigma$-loop $\gamma = \{[(0,0),2],[(1,0),2],[(1,1),1],[(0,1),1],[(0,0),2]\}$ has a nonzero image vector $\omega_\sigma(\gamma) = (1,-1)+(0,1)-(1,0)-(0,1) = (0,-1)$, so σ in inconsistent by Proposition 2.2. This is also illustrated by the following:

Example 2.5. The substitution defined by the base rule $1 \mapsto \{[(0,0),1\}$ and the concatenation rules $\boxed{1}\boxed{1} \mapsto \boxed{1}\boxed{1}$, $\begin{smallmatrix}1\\1\end{smallmatrix} \mapsto \boxed{1}\boxed{1}$ is overlapping: the images of the cells $[(0,1),1]$ and $[(1,0),1]$ overlap in the image of the pattern $\{[(0,0),1],[(1,0),1],[(0,1),1]\}$:



3 Undecidability Results

Wang tiles are unit square tiles with colored edges, which are oriented and may not be rotated. We say that a set of Wang tiles T admits a valid tiling of a cycle if there exists a nontrivial sequence $(a_1,\dots,a_n)$ of translates of tiles of T such that a_i and a_{i+1} share exactly one edge and their colors agree on it for all $1 \leqslant i < n$, and such that $a_n = a_1$ (the other tiles a_i cannot overlap and are distinct). Because we require the cycle to be nontrivial, we must have $n \geqslant 5$.

In [7], it is proved that the following problem is undecidable: "Does a given finite set of Wang tiles admit a valid tiling of a cycle?" This problem is called the *weak cycle tiling problem* in [7]. (The strong version of the same problem requires that *any* two adjacent tiles in the cycle match in color, and not only a_i and a_{i+1}.) We will use the fact that this problem is undecidable in order to prove Theorems 3.1 and 3.3.

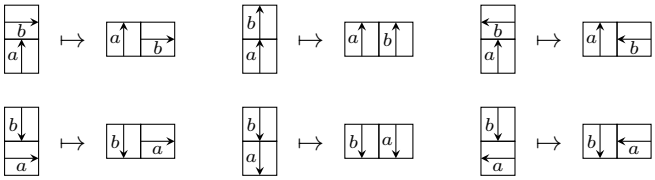
The undecidability results below are proved for two-dimensional substitutions. The proofs can easily be modified to get undecidability in higher dimensions, but dimension 1 has to be ruled out.

3.1 Undecidability of Consistency

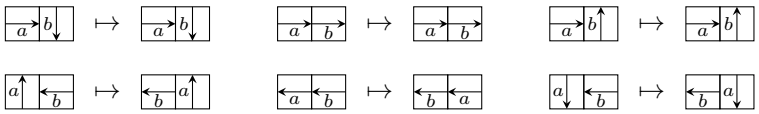
Theorem 3.1. *It is undecidable whether a substitution is consistent.*

Proof. We are going to reduce the cycle tiling problem for Wang tiles to the consistency problem for substitutions. Since the former is undecidable, the result will follow.

Let T be a set of Wang tiles. Let $\mathcal{A} = T \times \{\rightarrow, \uparrow, \leftarrow, \downarrow\}$, and σ be the substitution over alphabet $\mathcal{A}$ defined by $\sigma_{\text{base}}(t) = [(0,0), t]$ for all $t \in \mathcal{A}$, with the rules



for every tiles $a, b \in T$ such that the edges match in $\begin{array}{|c|}\hline b \\ \hline a \\ \hline \end{array}$, and the rules



for every tiles $a, b \in T$ such that the edges match in $\boxed{a}\boxed{b}$. By definition of the above rules, the set $\mathcal{C}_\sigma$ consists of all the valid dominoes of tiles of T, where in each domino, exactly one of the two tiles points at the other. The image of a domino by σ_{base} is then the concatenation of the pointing tile and the pointed tile, from left to right respectively.

We can now finish the proof by showing that T admits a valid tiling of a cycle if and only if σ is not consistent. Indeed, suppose that T admits a valid tiling of a cycle $(a_1, \ldots, a_n)$. To this cycle corresponds a $\mathcal{C}_\sigma$-loop $\gamma = (c_1, \ldots, c_n)$ where the type of each c_i is (a_i, d_i) and the arrow d_i points at the cell c_{i+1}, for $1 \leqslant i < n$ (and d_n points at c_1). However, we have $\omega_\sigma(\gamma) = (n-1, 0) \neq (0,0)$, so σ is not consistent, by Proposition 2.2.

Conversely, if T does not admit a valid tiling of a cycle then there cannot exist any simple $\mathcal{C}_\sigma$-loop, so σ is consistent thanks to Proposition 2.2. □

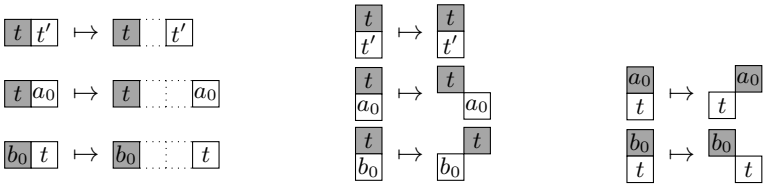
Remark 3.2. The above proof yields a stronger version of Theorem 3.1: consistency is undecidable for two-dimensional domino-to-domino substitutions. It can be proved that this undecidability result also holds for non-overlapping substitutions.

3.2 Undecidability of Overlapping

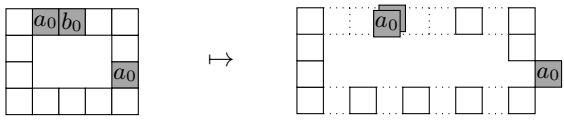
The weak cycle tiling problem can also be used to prove the undecidability of overlapping for consistent two-dimensional substitutions.

Theorem 3.3. *It is undecidable whether a consistent substitution is overlapping.*

Proof. We will reduce the cycle tiling problem. Let T be a set of Wang tiles. Given two tiles $a, b \in T$ whose colors match in $\boxed{a}\boxed{b}$, let $\sigma_{a,b}$ be the two-dimensional substitution defined on the alphabet $\mathcal{A} = T \cup \{a_0, b_0\}$ where a_0 and b_0 are two new states, with the base rule $t \mapsto \{[(0,0), t]\}$ for all $t \in \mathcal{A}$ and the concatenation rules



for the $t, t' \in \mathcal{A} \setminus \{a_0, b_0\}$ such that the tiles match in left-hand sides of the above rules (we require a_0 and b_0 to match in the same way as a and b). On patterns without a_0 or b_0, the image of a pattern by $\sigma_{a,b}$ is a copy expanded horizontally by a factor of two (leaving one horizontal gap between horizontal neighbors). When a_0 or b_0 is in the pattern, the action of $\sigma_{a,b}$ is the same, and in addition it shifts every occurrence of a_0 to the right and every occurrence of b_0 to the left, as illustrated below. Note that the images of a_0 and b_0 are shifted but that the images of a and b are not (they are treated like the other cells).



The rule is hence consistent, and an overlap can happen only between the images of a cell of type a_0 and a cell of type b_0: an overlap occurs if and only if the image of $\boxed{a_0}\boxed{b_0}$ is computed (as shown in the above picture). It follows that $\sigma_{a,b}$ is overlapping if and only if there exists a Wang tile cycle of T that contains $\boxed{a}\boxed{b}$. Indeed, if there exists such a cycle, then the corresponding pattern in which exactly one occurrence of $\boxed{a}\boxed{b}$ is replaced by $\boxed{a_0}\boxed{b_0}$ can be computed (is $\mathcal{C}_{\sigma_{a,b}}$-covered) and will cause an overlap. Conversely, if an overlap exists then we know that it is because the image of $\boxed{a_0}\boxed{b_0}$ has been computed. This is possible only if a cycle of T containing $\boxed{a}\boxed{b}$ exists, because $\boxed{a_0}\boxed{b_0}$ is not a starting pattern of $\sigma_{a,b}$.

Now we can finish the reduction. Given a set of Wang tiles T, compute $\sigma_{a,b}$ for every tiles a, b whose colors match in $\boxed{a}\boxed{b}$. One of the substitutions $\sigma_{a,b}$ is overlapping if and only if T admits a tiling of a cycle. □

Remark 3.4. The above proof yields a stronger version of Theorem 3.3: non-overlapping is undecidable for consistent two-dimensional domino substitutions. One can also prove that this holds even for domino-to-domino substitutions.

4 Decidability Results

In this section we give algorithms to decide the consistency or non-overlapping of a natural class of substitutions: the substitutions σ that are *domino-complete*,

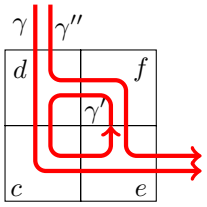
that is, such that the set of starting patterns $\mathcal{C}_\sigma$ is the set of *all* the possible dominoes.

4.1 Decidability of Consistency for Domino-Complete Substitutions

Theorem 4.1. *It is decidable whether a given two-dimensional domino-complete substitution is consistent. More precisely, such a substitution is consistent if and only if it is consistent on every* 2×2 *pattern.*

Proof. The "only if" implication is trivial. For the "if" implication, suppose that σ is not consistent. By Proposition 2.2, there exists a simple $\mathcal{C}_\sigma$-loop $\gamma = (c_1, \ldots, c_n)$ such that $\omega_\sigma(\gamma) \neq 0$. We will prove that there exists a 2×2 pattern on which σ is not consistent by "reducing" γ inductively.

Let c be the lowest cell on the leftmost column of γ. Since γ does not overlap itself, there exist two cells $d, e \in \gamma$ such that d is above c and e is at the right of c. We suppose, without loss of generality, that d, c, e appear in this order in γ, *i.e.*, $\gamma = (c_1, \ldots, c_i, d, c, e, c_{i+4}, \ldots, c_n)$. Let $f = [v + (1,1), t]$, where v is the vector of c and $t \in \mathcal{A}$ is arbitrary (or t agrees with γ if γ already contains a cell of vector $v + (1,1)$). Let $\gamma' = (c, e, f, d, c)$ and $\gamma'' = (c_1, \ldots, c_i, d, f, e, c_{i+4}, \ldots, c_n)$, as shown below.



We have $\omega_\sigma(\gamma') + \omega_\sigma(\gamma'') = \omega_\sigma(\gamma) \neq 0$, so $\omega_\sigma(\gamma') \neq 0$ or $\omega_\sigma(\gamma'') \neq 0$, which implies the existence of a $\mathcal{C}_\sigma$-loop (γ' or γ'') with nonzero image vector which surrounds strictly less cells than γ (unless γ consists 4 cells already). Now, in the same way as in the second part of the proof of Proposition 2.2, γ' or γ'' must contain a simple loop with nonzero image vector. Applying this reasoning inductively eventually leads to a 2×2 loop γ such that $\omega_\sigma(\gamma) \neq 0$, which concludes the proof. □

Generalization to Domino-Completeness within a Set of Patterns. We now want to generalize Theorem 4.1 to substitutions that are domino-complete only within a particular set of patterns. Let us first state a few definitions. If $\mathcal{P}$ is a set of patterns, we say that σ is *$\mathcal{P}$-domino-complete* if $\mathcal{C}_\sigma$ is equal to the set of dominoes that appear in the patterns of $\mathcal{P}$. If $\mathcal{S} \subseteq \mathcal{A}^{\mathbb{Z}^2}$, we denote by $\mathrm{patt}(\mathcal{S})$ the set of the patterns that appear in the elements of $\mathcal{S}$. (That is, a pattern is in $\mathrm{patt}(\mathcal{S})$ if it can be extended to an element of $\mathcal{S}$.)

Theorem 4.2 below gives a simple criterion to determine if a $\mathcal{P}$-domino-complete substitution is consistent when $\mathcal{P}$ is the set of all the 2×2 patterns of some $\mathcal{S} \subseteq \mathcal{A}^{\mathbb{Z}^2}$. Note that to decide this property, we must be able to compute

the 2×2 patterns of $\mathcal{S}$, which is not necessarily possible. Note that Theorem 4.1 can be seen as the particular case of Theorem 4.2 when $\mathcal{S} = \mathcal{A}^{\mathbb{Z}^2}$.

Theorem 4.2. *Let $\mathcal{S} \subseteq \mathcal{A}^{\mathbb{Z}^2}$ and let $\mathcal{P} = \{P_1, \dots, P_n\}$ be the list of the 2×2 patterns that appear in the elements of $\mathcal{S}$. A $\mathcal{P}$-domino-complete substitution is consistent on* $\mathrm{patt}(\mathcal{S})$ *if and only if it is consistent on the patterns $P_1, \dots, P_n$.*

Proof. Let P be a pattern of $\mathrm{patt}(\mathcal{S})$ that contains a $\mathcal{C}_\sigma$-loop γ such that $\omega_\sigma(\gamma) \neq 0$. We cannot directly reduce the loop as in the proof of Theorem 4.1 because σ is not domino-complete. However, σ is $\mathcal{P}$-domino complete so there exists $c \in \mathcal{S}$ that contains P. We can then reduce γ within c to a 2×2 loop, as explained in the proof of Theorem 4.1. It follows that σ is consistent on $\mathrm{patt}(\mathcal{S})$ if and only if it is consistent on $P_1, \dots, P_n$. □

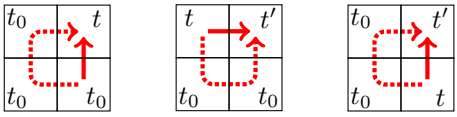
4.2 Decidability of Overlapping for Consistent Domino-Complete Substitutions

We now focus on the domino-complete substitutions that are consistent. Proposition 4.3 below tells us that such substitutions are simple: there exists $\alpha, \beta \in \mathbb{Z}^2$ such that the image of the cell placed at (x, y) is placed at $(x\alpha, y\beta)$ in the lattice $\alpha\mathbb{Z} \times \beta\mathbb{Z}$. We will use this proposition to give an algorithm that decides if a consistent domino-complete is overlapping (Theorem 4.4).

Unfortunately, we are not able to give an analogue of Theorem 4.2 for the non-overlapping property, because the associated decision problems seem to become too difficult to track.

Proposition 4.3. *Let σ be a consistent two-dimensional domino-complete substitution. There exist two vectors $\alpha, \beta \in \mathbb{Z}^2$ and a vector $v_t \in \mathbb{Z}^2$ for every $t \in \mathcal{A}$ such that for every $\mathcal{C}_\sigma$-path γ from a cell $[(0,0), t]$ to a cell $[(x, y), t']$, we have $\omega_\sigma(\gamma) = x\alpha + y\beta - v_t + v_{t'}$. Moreover, α, β and v_t can be obtained effectively.*

Proof. Let $t_0 \in \mathcal{A}$ be arbitrary, where $\mathcal{A}$ is the alphabet of σ. Let $\alpha = \omega_\sigma(\gamma)$ and $\beta = \omega_\sigma(\gamma')$, where $\gamma = ([(0,0), t_0], [(1,0), t_0])$ and $\gamma' = ([(0,0), t_0], [(0,1), t_0])$. For $t \in \mathcal{A}$, define $v_t = \omega_\sigma(\gamma) - \alpha$, where $\gamma = ([(0,0), t_0], [(1,0), t])$. We first prove the theorem for paths of length two. Because σ is consistent and domino-complete, we can use the following patterns



to compute the values

$$\begin{aligned}
\omega_\sigma([(0,0), t_0], [(0,1), t]) &= -\alpha + \beta + \alpha + v_t &&= \beta + v_t \\
\omega_\sigma([(0,0), t], [(1,0), t']) &= -\beta - v_t + \alpha + \beta + v_{t'} &&= \alpha - v_t + v_{t'} \\
\omega_\sigma([(0,0), t], [(0,1), t']) &= -\alpha - v_t + \beta + \alpha + v_{t'} &&= \beta - v_t + v_{t'}
\end{aligned}$$

which proves the statement for paths of length two. The statement for arbitrary paths follows directly, by adding the consecutive dominoes along the path. □

Theorem 4.4. *It is decidable whether a given two-dimensional consistent domino-complete substitution σ is overlapping.*

Proof. By Proposition 4.3, we can compute $\alpha, \beta, v_t, v_{t'} \in \mathbb{Z}^2$ such that $\omega_\sigma(\gamma) = x\alpha + y\beta - v_t + v_{t'}$ for every $\mathcal{C}_\sigma$-path γ from a cell $[(0,0),t]$ to a cell $[(x,y),t']$.

Denote $A_t = \mathrm{supp}(\sigma_{\mathrm{base}}(t))$ for $t \in \mathcal{A}$. By definition, σ is overlapping if and only if there exists $t, t' \in \mathcal{A}$ and $x, y \in \mathbb{Z}$ such that $A_t \cap \{b + x\alpha + y\beta - v_t + v_{t'} : b \in A_{t'}\} \neq \varnothing$. This leaves a finite number of linear equations to check: for each $(t,t') \in \mathcal{A}^2$, we check if there exists $a \in A_t$ and $b \in A_{t'}$ such that the following equation has a nonzero solution $(x,y) \in \mathbb{Z}^2$:

$$a \;=\; b + x\alpha + y\beta - v_t + v_{t'}$$

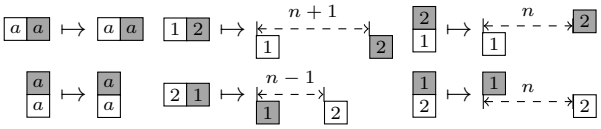
This can be done algorithmically and σ is overlapping if and only if such a solution exists. □

Example 4.5. Let σ be the two-dimensional substitution on the alphabet $\{1\}$ defined by the base rule

$$1 \;\mapsto\; \{[(0,0),1],[(1,0),1],[(2,0),1],[(1,1),1]\} \;=\; \begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

and the concatenation rules $(1,1,(1,0)) \mapsto (3,0)$ and $(1,1,(0,1)) \mapsto (0,2)$. This substitution is domino-complete and consistent. Proposition 4.3 applied to σ gives $\omega_\sigma(\gamma) = (3x,0) + (0,2y)$ for every $\mathcal{C}_\sigma$-path γ from a cell $[(0,0),1]$ to a cell $[(x,y),1]$. Note that in this case, $v_1 = (0,0)$.

Example 4.6. For $n \geqslant 0$, let σ_n be the two-dimensional substitution on the alphabet $\{1,2\}$ defined by the base rule $1 \mapsto \{[(0,0),1]\}$, $2 \mapsto \{[(0,0),2]\}$ and the concatenation rules



The substitution σ_n is domino-complete and consistent for all n. Proposition 4.3 applied to σ_n gives $\omega_\sigma(\gamma) = (x,y) - v_t + v_{t'}$ for every $\mathcal{C}_\sigma$-path γ from a cell $[(0,0),t]$ to a cell $[(x,y),t']$, where $v_1 = (0,0)$ and $v_2 = (n,0)$. This gives an example of a substitution with at least one nonzero v_t.

This example is also interesting because it is overlapping, but only on sufficiently large patterns. Indeed, it is non-overlapping on patterns of horizontal diameter smaller than n, but overlapping on larger patterns such as

$$P_n \;=\; \begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 1 & \cdots & 1 & 1 \\ \hline 2 & & & & & 1 \\ \hline \end{array}\;\; \overset{n}{\longleftarrow\!\!\longrightarrow} \;=\; [(0,0),2] \cup [(n+1,0),1] \cup \{[(i,1),1] : 0 \leqslant i \leqslant n+1\}.$$

This shows that the overlapping property cannot be decided as simply as consistency, where looking at the 2×2 patterns was sufficient. The size of the rules has to be taken into account.

5 Conclusion and Open Problems

In this article we focused on the consistency and the overlapping of substitutions, because they are the first properties that one should look at, in order to make sure that the substitution is "well defined". There are many other interesting properties whose decidability status has not been investigated yet. Some of them are:

- The overlapping property in the consistent and "weakened" domino-complete case, as we have done for consistency in Theorem 4.2.
- Iterability: given a substitution σ and a pattern P, can σ be iterated on the successive images of P by σ? (That is, are the successive images of P by σ all $\mathcal{C}_\sigma$-covered?)
- Consistency in the primitive case. A substitution is *primitive* if for every cell of any type, there exists $n \in \mathbb{N}$ such that the pattern $\sigma^n(\{c\})$ contain cells of *every* type of $\mathcal{A}$. (This presupposes that the substitution can be iterated.) Is it decidable if a primitive substitution is consistent?
- (Simple) connectedness: given a substitution σ and a pattern P, are the iterates of σ on P (simply) connected?
- The "growing squares" property: given a substitution σ and a pattern P, do the iterates of σ on P contain arbitrarily large squares?

Acknowledgements. Many thanks to Thomas Fernique for helpful discussions. Research supported by the Academy of Finland Grant 131558.

References

1. Pytheas Fogg, N.: Substitutions in dynamics, arithmetics and combinatorics. Lecture Notes in Mathematics, vol. 1794. Springer, Berlin (2002)
2. Frank, N.P.: A primer of substitution tilings of the Euclidean plane. Expo. Math. 26, 295–326 (2008)
3. Fernique, T.: Local rule substitutions and stepped surfaces. Theoret. Comput. Sci. 380, 317–329 (2007)
4. Arnoux, P., Berthé, V., Siegel, A.: Two-dimensional iterated morphisms and discrete planes. Theoret. Comput. Sci. 319, 145–176 (2004)
5. Frank, N.P.: Detecting combinatorial hierarchy in tilings using derived Voronoï tesselations. Discrete Comput. Geom. 29, 459–476 (2003)
6. Jolivet, T., Kari, J.: Consistency of multidimensional combinatorial substitutions. Theoret. Comput. Sci. (to appear, 2012)
7. Kari, J.: Infinite Snake Tiling Problems. In: Ito, M., Toyama, M. (eds.) DLT 2002. LNCS, vol. 2450, pp. 67–77. Springer, Heidelberg (2003)

Two-Way Automata Characterizations of L/poly versus NL

Christos A. Kapoutsis[1,⋆] and Giovanni Pighizzini[2]

[1] LIAFA, Université Paris VII, France
[2] DICo, Università degli Studi di Milano, Italia

Abstract. Let $\mathsf{L/poly}$ and NL be the standard complexity classes, of languages recognizable in logarithmic space by Turing machines which are deterministic with polynomially-long advice and nondeterministic without advice, respectively. We recast the question whether $\mathsf{L/poly} \supseteq \mathsf{NL}$ in terms of deterministic and nondeterministic two-way finite automata (2DFAs and 2NFAs). We prove it equivalent to the question whether every s-state *unary* 2NFA has an equivalent poly(s)-state 2DFA, or whether a poly(h)-state 2DFA can check accessibility in h-vertex graphs (even under unary encoding) or check two-way liveness in h-tall, h-column graphs. This complements two recent improvements of an old theorem of Berman and Lingas. On the way, we introduce new types of reductions between regular languages (even unary ones), use them to prove the completeness of specific languages for two-way nondeterministic polynomial size, and propose a purely combinatorial conjecture that implies $\mathsf{L/poly} \not\supseteq \mathsf{NL}$.

1 Introduction

A prominent open question in complexity theory asks whether nondeterminism is essential in logarithmic-space Turing machines. Formally, this is the question whether $\mathsf{L} = \mathsf{NL}$, for L and NL the standard classes of languages recognizable by logarithmic-space deterministic and nondeterministic Turing machines.

In the late 70's, Berman and Lingas [1] connected this question to the comparison between deterministic and nondeterministic two-way finite automata (2DFAs and 2NFAs), proving that *if* $\mathsf{L} = \mathsf{NL}$*, then for every* s*-state* σ*-symbol* 2NFA *there is a* poly($s\sigma$)*-state* 2DFA *which agrees with it on all inputs of length* $\leq s$. (They also proved that this implication becomes an equivalence if we require that the 2DFA be constructible from the 2NFA in logarithmic space.)

Recently, two improvements of this theorem have been announced. On the one hand, Geffert and Pighizzini [5] proved that *if* $\mathsf{L} = \mathsf{NL}$ *then for every* s*-state unary* 2NFA *there is an equivalent* poly(s)*-state* 2DFA. That is, in the special case where $\sigma = 1$, the Berman-Lingas theorem becomes true without any restriction on input lengths. On the other hand, Kapoutsis [7] proved that $\mathsf{L/poly} \supseteq \mathsf{NL}$ *iff for every* s*-state* σ*-symbol* 2NFA *there is a* poly(s)*-state* 2DFA *which agrees*

⋆ Supported by a Marie Curie Intra-European Fellowship (PIEF-GA-2009-253368) within the European Union Seventh Framework Programme (FP7/2007-2013).

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 217–228, 2012.

with it on all inputs of length $\leq s$, where $\mathsf{L/poly}$ is the standard class of languages recognizable by deterministic logarithmic-space Turing machines with polynomially-long advice. Hence, the Berman-Lingas theorem is true even under the weaker assumption $\mathsf{L/poly} \supseteq \mathsf{NL}$, even for the stronger conclusion where the 2DFA size is independent of σ, and then even in the converse direction.

A natural question arising from these developments is whether the theorem of [5] can be strengthened to resemble that of [7]: Does the implication remain true under the weaker assumption that $\mathsf{L/poly} \supseteq \mathsf{NL}$? If so, does it then become an equivalence? Indeed, we prove that $\mathsf{L/poly} \supseteq \mathsf{NL}$ *iff for every* s*-state unary* 2NFA *there is an equivalent* poly(s)*-state* 2DFA. Intuitively, this means that $\mathsf{L/poly} \supseteq \mathsf{NL}$ is true not only iff 'small' 2NFAs can be simulated by 'small' 2DFAs on *'short' inputs* (as in [7]), but also iff the same is true for *unary inputs*.

In this light, a second natural question is whether this common behavior of 'short' and unary inputs is accidental or follows from deeper common properties. Indeed, our analysis reveals two such properties. They are related to *outer-nondeterministic* 2FAs (2OFAs, which perform nondeterministic choices only on the end-markers [2]) and to the *graph accessibility problem* (GAP, the problem of checking the existence of paths in directed graphs), and use the fact that checking whether a 'small' 2OFA M accepts an input x reduces to solving GAP in a 'small' graph $G_M(x)$. The first common property is that, both on 'short' and on unary inputs, 'small' 2NFAs can be simulated by 'small' 2OFAs (Lemma 1). The second common property is that, both on 'short' and on unary inputs, it is possible to encode instances of GAP so that a 'small' 2DFA can extract $G_M(x)$ from x (Lemma 5) and simultaneously simulate on it another 'small' 2DFA (Lemma 6). For 'short' inputs, both properties follow from standard ideas; for unary inputs, they follow from the analyses of [3,8] and a special unary encoding for graphs.

We work in the complexity-theoretic framework of [9]. We focus on the class 2D of (families of promise) problems solvable by polynomial-size 2DFAs, and on the corresponding classes $\mathsf{2N/poly}$ and $\mathsf{2N/unary}$ for 2NFAs and for problems with polynomially-long and with unary instances, respectively. In these terms, $\mathsf{2D} \supseteq \mathsf{2N/poly}$ means 'small' 2DFAs can simulate 'small' 2NFAs on *'short' inputs*; $\mathsf{2D} \supseteq \mathsf{2N/unary}$ means the same for *unary inputs*; the theorem of [7] is that $\mathsf{L/poly} \supseteq \mathsf{NL} \Leftrightarrow \mathsf{2D} \supseteq \mathsf{2N/poly}$; the theorem of [5] is the forward implication that $\mathsf{L} \supseteq \mathsf{NL} \Rightarrow \mathsf{2D} \supseteq \mathsf{2N/unary}$; and our main contribution is the stronger statement

$$\mathsf{L/poly} \supseteq \mathsf{NL} \iff \mathsf{2D} \supseteq \mathsf{2N/unary}\,. \tag{1}$$

This we derive from [7] and the equivalence $\mathsf{2D} \supseteq \mathsf{2N/poly} \Leftrightarrow \mathsf{2D} \supseteq \mathsf{2N/unary}$, obtained by our analysis of the common properties or 'short' and unary inputs.

Our approach returns several by-products of independent interest, already anticipated in [6] for enriching the framework of [9]: new types of reductions, based on 'small' two-way deterministic finite transducers; the completeness of binary and unary versions of GAP (BGAP and UGAP) for $\mathsf{2N/poly}$ and $\mathsf{2N/unary}$, respectively, under such reductions (Cor. 2); the closure of 2D under such reductions (Cor. 3); and the realization of the central role of 2OFAs in this framework (as also recognized in [2]). In the end, our main theorem (Th. 1) is the equivalence

of $\mathsf{L/poly} \supseteq \mathsf{NL}$ to all these statements (and one more):

$$\mathsf{2D} \supseteq \mathsf{2N/poly} \quad \mathsf{2D} \supseteq \mathsf{2N/unary} \quad \mathsf{2D} \supseteq \mathsf{2O} \quad \mathsf{2D} \ni \text{BGAP} \quad \mathsf{2D} \ni \text{UGAP}$$

where 2O is the analogue of 2D for 2OFAs. Hence, the conjecture $\mathsf{L/poly} \not\supseteq \mathsf{NL}$ is now the conjecture that all these statements are false. In this context, we also propose a stronger conjecture, both in algorithmic and in combinatorial terms.

Concluding this introduction, we note that, of course, (1) can also be derived by a direct proof of each direction. In such a derivation, the forward implication is a straightforward strengthening of the proof of [5], but the backward implication needs the encoding of UGAP and the ideas behind Lemma 6.2.

2 Preparation

If $h \geq 1$, we let $[h]{:=}\{0, 1, \ldots, h{-}1\}$, use K_h for the complete directed graph with vertex set $[h]$, and p_h for the h-th smallest prime number. If x is a string, then $|x|$, x_i, and x^i are its length, its i-th symbol ($1 \leq i \leq |x|$), and the concatenation of i copies of it ($i \geq 0$). The empty string is denoted by ϵ.

The *binary encoding* of a subgraph G of K_h, denoted as $\langle G \rangle_2$, is the standard h^2-bit encoding of the adjacency matrix of G: arrow (i, j) is present in G iff the $(i{\cdot}h{+}j{+}1)$-th most significant bit of the encoding is 1. The *unary encoding* of G, denoted as $\langle G \rangle_1$, uses the natural correspondence between the bits of $\langle G \rangle_2$ and the h^2 smallest prime numbers, where the k-th most significant bit maps to p_k, and thus arrow (i, j) maps to the prime number $p_{(i,j)}{:=}p_{i \cdot h+j+1}$. We let $\langle G \rangle_1{:=}0^{n_G}$, where the length n_G is the product of the primes which correspond to the 1s of $\langle G \rangle_2$, and thus to the arrows of K_h which are present in G:

$$n_G := \prod_{\text{bit } k \text{ of } \langle G \rangle_2 \text{ is } 1} p_k \quad = \prod_{(i,j) \text{ is in } G} p_{(i,j)} \quad = \prod_{(i,j) \text{ is in } G} p_{i \cdot h+j+1} \ . \tag{2}$$

Note that, conversely, every length $n \geq 1$ determines the unique subgraph $K_h(n)$ of K_h where each arrow (i, j) is present iff the corresponding prime $p_{(i,j)}$ divides n. Easily, $G = K_h(n_G)$.

A *prime encoding* of a length $n \geq 1$ is any string $\#z_1\#z_2\#\cdots\#z_m \in (\#\Sigma^*)^*$, where $\# \notin \Sigma$ and each z_i encodes one of the m prime powers in the prime factorization of n. Here, the alphabet Σ and the encoding scheme for the z_i are arbitrary, but fixed; the order of the z_i is arbitrary.

A (*promise*) *problem* over Σ is a pair $\mathfrak{L} = (L, \tilde{L})$ of disjoint subsets of Σ^*. An *instance* of $\mathfrak{L}$ is any $x \in L \cup \tilde{L}$. If $\tilde{L} = \Sigma^* - L$ then $\mathfrak{L}$ is a *language*. If $\mathcal{L} = (\mathfrak{L}_h)_{h \geq 1}$ is a family of problems, its members are *short* if every instance of every $\mathfrak{L}_h$ has length $\leq p(h)$, for some polynomial p. If $\mathcal{M} = (M_h)_{h \geq 1}$ is a family of machines, its members are *small* if every M_h has $\leq p(h)$ states, for some polynomial p.

2.1 Automata

A *two-way nondeterministic finite automaton* (2NFA) consists of a finite control and an end-marked, read-only tape, accessed via a two-way head. Formally, it is a

tuple $M = (S, \Sigma, \delta, q_0, q_f)$ of a set of states S, an alphabet Σ, a start state $q_0 \in S$, a final state $q_f \in S$, and a set of transitions $\delta \subseteq S \times (\Sigma \cup \{\vdash, \dashv\}) \times S \times \{\text{L,R}\}$, for $\vdash, \dashv \notin \Sigma$ two end-markers and L:=−1 and R:=+1 the two directions. An input $x \in \Sigma^*$ is presented on the tape as $\vdash x \dashv$. The computation starts at q_0 on $\vdash$. At each step, M uses δ on the current state and symbol to decide the possible next state-move pairs. We assume δ never violates an end-marker, except if on $\dashv$ and the next state is q_f. A *configuration* is a state-position pair in $S \times [\,|x|+2\,]$ or the pair $(q_f, |x|+2)$. The computation produces a tree of configurations, and x is *accepted* if some branch 'falls off $\dashv$ into q_f', i.e., ends in $(q_f, |x|+2)$. A problem $(L, \tilde{L})$ is *solved* by M if M accepts all $x \in L$ but no $x \in \tilde{L}$.

We say M is *outer-nondeterministic* (2OFA [2]) if all nondeterministic choices are made on the end-markers: formally, for each $(q, a) \in S \times \Sigma$ there is at most one transition of the form $(q, a, \,.\,, \,.\,)$; if this holds also for each $(q, a) \in S \times \{\vdash, \dashv\}$, then M is *deterministic* (2DFA). We say M is *sweeping* (SNFA, SOFA, SDFA) if the head reverses only on the end-markers: formally, the set of transitions is now just $\delta \subseteq S \times (\Sigma \cup \{\vdash, \dashv\}) \times S$ and the next position is defined to be always the adjacent one in the direction of motion; except if the head is on $\vdash$ or if the head is on $\dashv$ and the next state is not q_f, in which two cases the head reverses. We say M is *rotating* (RNFA, ROFA, RDFA) if it is sweeping, but modified so that its head jumps directly back to $\vdash$ every time it reads $\dashv$ and the next state is not q_f: formally, the next position is now always the adjacent one to the right; except when the head is on $\dashv$ and the next state is not q_f, in which case the next position is on $\vdash$. Restricting the general definition of outer-nondeterminism, we require that a ROFA makes all nondeterministic choices exclusively on $\vdash$.

Lemma 1. *For every s-state* 2NFA *M and length bound l, there is a $O(s^2 l^2)$-state* ROFA *M' that agrees with M on all inputs of length $\leq l$. If M is unary, then M' has $O(s^2)$ states and agrees with M on all inputs (of any length).*

Proof. Pick any 2NFA $M = (S, \Sigma, \delta, q_0, q_f)$ and length bound l. To simulate M on inputs x of length $n \leq l$, a ROFA $M' = \big(S', \Sigma, \,.\,, (q_0, 0), q'_f\big)$ uses the states

$$S' := \big(S \times [l+2]\big) \cup \big(S \times [l+2] \times S \times \{\text{L,R}\} \times [l+1]\big) \cup \{\, q'_f \,\}.$$

The event of M being in state q and on tape cell i (where $0 \leq i \leq n+1$, cell 0 contains $x_0 := \vdash$ and cell $n+1$ contains $x_{n+1} := \dashv$) is simulated by M' being in state (q, i) and on $\vdash$. From there, M' nondeterministically 'guesses' among all $(p, d) \in S \times \{\text{L,R}\}$ a 'correct' next transition of M out of q and x_i (i.e., a transition leading to acceptance) and goes on to verify that this transition is indeed valid, by deterministically sweeping the input up to cell i (using states of the form (q, i, p, d, j) with $j < i$) and checking that indeed $(q, x_i, p, d) \in \delta$. If the check fails, then M' just hangs (in this branch of the computation). Otherwise, it continues the sweep in state $(p, i + d)$ until it reaches $\dashv$ and jumps back to $\vdash$; except if $x_i = \dashv$ & $p = q_f$ & $d = \text{R}$, in which case it just falls off $\dashv$ into q'_f.

If M is unary, then it can be simulated by a SOFA $\tilde{M}$ with $\tilde{s} = O(s^2)$ states [3]. This can then be converted into a ROFA M' with $2\tilde{s} + 1$ states, which simply replaces backward sweeps with forward ones—a straightforward simulation, since reversing a unary input does not change it. Hence, M' is also of size $O(s^2)$. □

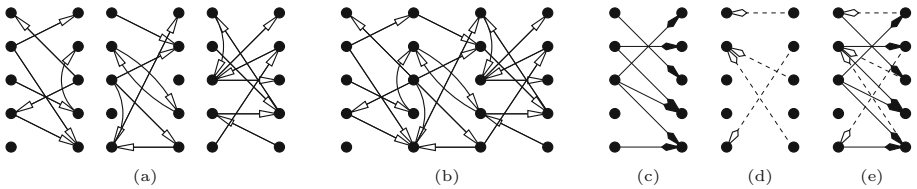


Fig. 1. (a) Three symbols of Γ_5. (b) Their string as a multi-column graph. (c) A symbol $(A,\text{L}) \in \Delta_5$. (d) A symbol $(B,\text{R}) \in \Delta'_5$. (e) The overlay of the symbols of (c) and (d).

A family of 2NFAs $\mathcal{M} = (M_h)_{h\geq 1}$ *solves* a family of problems $\mathcal{L} = (\mathfrak{L}_h)_{h\geq 1}$ if every M_h solves $\mathfrak{L}_h$. The class of families of problems that admit small 2NFAs is

$$\mathsf{2N} := \{\mathcal{L} \mid \mathcal{L} \text{ is a family of problems solvable by a family of } \textit{small} \text{ 2NFAs}\}.$$

Its restrictions to families of *short* and of *unary* problems are respectively $\mathsf{2N/poly}$ and $\mathsf{2N/unary}$. Analogous classes for restricted 2NFAs are named similarly: e.g., the class for problems with small 2DFAs is $\mathsf{2D}$, the class for short problems with small ROFAs is $\mathsf{RO/poly}$, etc.

Corollary 1. $\mathsf{2N/poly} = \mathsf{RO/poly}$ *and* $\mathsf{2N/unary} = \mathsf{RO/unary}$.

2.2 Problems

The *graph accessibility problem* on h vertices is: "Given a subgraph G of K_h, check that G contains a path from 0 to $h-1$." Depending on whether G is given in binary or unary, we get the following two formal variants of the problem:

$$\begin{aligned}
\text{BGAP}_h := \big(&\{\langle G\rangle_2 \mid G \text{ is subgraph of } K_h \text{ and has a path } 0 \rightsquigarrow h{-}1\},\\
&\{\langle G\rangle_2 \mid G \text{ is subgraph of } K_h \text{ and has no path } 0 \rightsquigarrow h{-}1\}\big)\\
\text{UGAP}_h := \big(&\{0^n \mid K_h(n) \text{ has a path } 0 \rightsquigarrow h{-}1\},\\
&\{0^n \mid K_h(n) \text{ has no path } 0 \rightsquigarrow h{-}1\}\big)
\end{aligned}$$

and the corresponding families $\text{BGAP}:=(\text{BGAP}_h)_{h\geq 1}$ and $\text{UGAP}:=(\text{UGAP}_h)_{h\geq 1}$.

Lemma 2. BGAP_h *and* UGAP_h *are solved by* ROFAs *with* $O(h^3)$ *and* $O(h^4 \log h)$ *states, respectively. Hence* $\text{BGAP} \in \mathsf{RO/poly}$ *and* $\text{UGAP} \in \mathsf{RO/unary}$.

The *two-way liveness* problem on height h, defined over the alphabet $\Gamma_h := \mathcal{P}(([h]\times\{\text{L,R}\})^2)$ of all h-tall directed two-column graphs (Fig. 1a), is: "Given a string x of such graphs, check that x is live." Here, $x \in \Gamma_h^*$ is *live* if the multi-column graph derived from x by identifying adjacent columns (Fig. 1b) contains 'live' paths, i.e., paths from the leftmost to the rightmost column; if not, then x is *dead*. Formally, this is the language $\text{TWL}_h:=\{x \in \Gamma_h^* \mid x \text{ is live}\}$. We focus

on two restrictions of this problem, in which x is promised to consist of $\leq h$ or of exactly 2 graphs. Formally, these are the promise problems and families

$$\textsc{short twl}_h := \big(\{x \in \Gamma_{\widetilde{h}}^{\leq h} \mid x \text{ is live}\},\ \{x \in \Gamma_{\widetilde{h}}^{\leq h} \mid x \text{ is dead}\} \big)$$
$$\textsc{compact twl}_h := \big(\{ab \in \Gamma_h^2 \mid ab \text{ is live}\},\ \{ab \in \Gamma_h^2 \mid ab \text{ is dead}\} \big)$$

and $\textsc{short twl}:=(\textsc{short twl}_h)_{h\geq 1}$, $\textsc{compact twl}:=(\textsc{compact twl}_h)_{h\geq 1}$.

Lemma 3. $\textsc{twl}_h$ *is solved by a* 2NFA *with* $2h$ *states. Hence* SHORT TWL *and* COMPACT TWL *are both in* RO/poly.

The *relational match* problem on $[h]$ is defined over the alphabet $\Delta_h := \mathbb{P}([h] \times [h]) \times \{\text{L},\text{R}\}$ of all pairs of binary relations on $[h]$ and side tags. A symbol (A,L) denotes an h-tall two-column graph with rightward arrows chosen by A (Fig. 1c); a symbol (B,R) denotes a similar graph with leftward arrows chosen by B (Fig. 1d). If the overlay of these two graphs (Fig. 1e) contains a cycle, we say that the symbols *match*, or just that A, B *match*. We consider the problem

$$\begin{aligned}\textsc{rm}_h := \big(&\{ (A,\text{L})(B,\text{R}) \mid A,B \subseteq [h]\times[h] \ \&\ A,B \text{ match}\},\\ &\{ (A,\text{L})(B,\text{R}) \mid A,B \subseteq [h]\times[h] \ \&\ A,B \text{ do not match}\} \big)\end{aligned}$$

of checking that two given relations match, and set $\textsc{rel match}:=(\textsc{rm}_h)_{h\geq 1}$. We also let $\textsc{fm}_h$ be the restriction of $\textsc{rm}_h$ to the alphabet $\Delta'_h := ([h] \rightharpoonup [h]) \times \{\text{L},\text{R}\}$, where all relations are partial functions, and set $\textsc{fun match}:=(\textsc{fm}_h)_{h\geq 1}$.

Lemma 4. $\textsc{fm}_h$ *is solved by a* 2DFA *with* h^3 *states. Hence* FUN MATCH $\in$ 2D.

Finally, $\textsc{rel zero-match} = (\textsc{rzm}_h)_{h\geq 1}$ and $\textsc{fun zero-match} = (\textsc{fzm}_h)_{h\geq 1}$ are the variants where we only check whether the given relations or functions '*match through* 0', i.e., create a cycle *through vertex* 0 *of the left column*.

3 Transducers, Reductions, and Completeness

A *two-way deterministic finite transducer* (2DFT) consists of a finite control, an end-marked, read-only input tape accessed via a two-way head, and an infinite, write-only output tape accessed via a one-way head. Formally, it is a tuple $T = (S, \Sigma, \Gamma, \delta, q_0, q_\text{f})$ of a set of states S, an input alphabet Σ, an output alphabet Γ, a start state $q_0 \in S$, a final state $q_\text{f} \in S$, and a transition function $\delta : S \times (\Sigma \cup \{\vdash,\dashv\}) \to S \times \{\text{L},\text{R}\} \times \Gamma^*$, where $\vdash,\dashv \notin \Sigma$. An input $x \in \Sigma^*$ is presented on the input tape as $\vdash x \dashv$. The computation starts at q_0 with the input head on $\vdash$ and the output tape empty. At every step, T applies δ on the current state and input symbol to decide the next state, the next input head move, and the next string to append to the contents of the output tape; if this string is non-empty, the step is called *printing*. We assume δ never violates an end-marker, except if the input head is on $\dashv$ and the next state is q_f. For $y \in \Gamma^*$, we say T *outputs* y and write $T(x) = y$, if T eventually falls off $\dashv$ and then the output tape contains y.

For $f : \Sigma^* \rightharpoonup \Gamma^*$ a partial function, we say T *computes* f if $T(x) = f(x)$ for all $x \in \Sigma^*$. If $\Gamma = \{\mathtt{0}\}$, then T can also 'compute' each unary string $f(x)$ by printing (not the string itself, but) an encoding of its length; if this is a prime encoding $\#z_1\#z_2\#\cdots\#z_m$ (cf. p. 219) and every infix $\#z_i$ is printed by a single printing step (possibly together with other infixes), then T *prime-computes* f.

Now let $\mathfrak{L} = (L, \tilde{L})$ and $\mathfrak{L}' = (L', \tilde{L}')$ be two problems over alphabets Σ and Σ'. A function $f : \Sigma^* \to (\Sigma')^*$ is a (*mapping*) *reduction* of $\mathfrak{L}$ to $\mathfrak{L}'$ if it is computable by a 2DFT and every $x \in \Sigma^*$ satisfies $x \in L \Rightarrow f(x) \in L'$ and $x \in \tilde{L} \Rightarrow f(x) \in \tilde{L}'$. If $\Sigma' = \{\mathtt{0}\}$ and f is prime-computable by a 2DFT, then f is a *prime reduction*. If $f(x)$ is always just $g(\vdash)g(x_1)\cdots g(x_{|x|})g(\dashv)$ for some homomorphism $g : \Sigma \cup \{\vdash,\dashv\} \to (\Sigma')^*$, then f is a *homomorphic reduction*. If such f exists, we say $\mathfrak{L}$ (*mapping-*)/*prime-*/*homomorphically reduces* to $\mathfrak{L}'$, and write $\mathfrak{L} \leq_{\mathrm{m}} \mathfrak{L}'$ / $\mathfrak{L} \preceq_{\mathrm{m}} \mathfrak{L}'$ / $\mathfrak{L} \leq_{\mathrm{h}} \mathfrak{L}'$. Easily, $\mathfrak{L} \leq_{\mathrm{h}} \mathfrak{L}'$ implies $\mathfrak{L} \leq_{\mathrm{m}} \mathfrak{L}'$.

Lemma 5. *If $\mathfrak{L}$ is solved by an s-state* 2OFA*, then $\mathfrak{L} \leq_{\mathrm{m}}$ BGAP_{2s+1} and $\mathfrak{L} \preceq_{\mathrm{m}}$ UGAP_{2s+1} and $\mathfrak{L} \leq_{\mathrm{h}}$ TWL_{2s}. The first two reductions are computable and prime-computable, respectively, by* 2DFTs *with $O(s^4)$ states and $O(s^2)$ printing steps per input; the last reduction maps strings of length n to strings of length $n+2$.*

Proof. Suppose $\mathfrak{L} = (L, \tilde{L})$ is solved by the s-state 2OFA $M = (S, \Sigma, ., q_0, q_{\mathrm{f}})$ and let $x \in \Sigma^*$. It is well-known that $\mathfrak{L} \leq_{\mathrm{h}}$ TWL_{2s} via a reduction f with $|f(x)| = |x|+2$ (e.g., see [7, Lemma 3]). So, we focus on the first two claims.

A *segment* of M on x is a computation of M on x that starts and ends on an end-marker visiting no end-markers in between, or a single-step computation that starts on $\dashv$ and falls off into q_{f}. The *end-points* of a segment are its first and last configurations. The *summary* of M on x is a subgraph $G(x)$ of K_{2s+1} that encodes all segments, as follows. The vertices represent segment end-points: vertex 0 represents $(q_0, 0)$; vertices $1, 2, \ldots, 2s-1$ represent the remaining points of the form $(q, 0)$ and all points of the form $(q, |x|+1)$; and vertex $2s$ represents $(q_{\mathrm{f}}, |x|+2)$. Hence, each arrow of K_{2s+1} represents a possible segment. The summary $G(x)$ contains exactly those arrows which correspond to segments that M can perform on x. Easily, every accepting branch in the computation of M on x corresponds to a path in $G(x)$ from vertex 0 to vertex $2s$, and vice-versa.

Now let the functions $f_2(x) := \langle G(x) \rangle_2$ and $f_1(x) := \langle G(x) \rangle_1$ map every x to an instance of BGAP_{2s+1} and of UGAP_{2s+1}. If $x \in L$, then the computation of M on x contains an accepting branch, so $G(x)$ contains a path $0 \rightsquigarrow 2s$, thus $f_2(x)$ and $f_1(x)$ are positive instances. If $x \in \tilde{L}$, then there is no accepting branch, hence $G(x)$ contains no path $0 \rightsquigarrow 2s$, thus $f_2(x)$ and $f_1(x)$ are negative instances. So, f_2 and f_1 are the desired reductions, if they can be computed appropriately.

To compute $f_2(x)$ from $x \in \Sigma^*$, a 2DFT T_2 iterates over all arrows of K_{2s+1}; for each of them, it checks whether M can perform on x the corresponding segment, and outputs $\mathtt{1}$ or $\mathtt{0}$ accordingly. To check a segment, from end-point (p, i) to end-point (q, j), it iterates over all configurations (p', i') that are nondeterministically visited by M right after (p, i); for each of them, it checks whether M can compute from (p', i') to (q, j); the segment check succeeds if any of these checks does. Finally, to check the computation from (p', i') to (q, j), the transducer could simulate M from (p', i') and up to the first visit to an end-marker. This is indeed

possible, since M would behave deterministically throughout this simulation. However, M could also loop, causing T_2 to loop as well, which is a problem.

To avoid this, T_2 simulates a halting variant M' of M, derived as follows. First, we remove from M all transitions performed on $\vdash$ or $\dashv$. Second, we add a fresh start state q'_0, along with transitions which guarantee that, on its first transition leaving q'_0, the machine will be in state p' and cell i' (since cell i' is adjacent to either $\vdash$ or $\dashv$, this requires either a single step from q'_0 to p' on $\vdash$ or a full forward sweep in q'_0 followed by a single backward step on $\dashv$ into p'). Third, we add a fresh final state q'_{f}, along with transitions which guarantee that, from state q reading the end-marker of cell j, the machine sweeps the tape in state q'_{f} until it falls off $\dashv$ (since cell j contains either $\vdash$ or $\dashv$, this is either a full forward sweep followed by a single step off $\dashv$, or just a single step off $\dashv$). Now we have a 2DFA which first brings itself into configuration (p', i'), then simulates M until the first visit to an end-marker, and eventually accepts only if this simulation reaches (q, j). So, this is a $(2{+}s)$-state 2DFA that accepts x iff M can perform on x the segment from (p', i') to (q, j). By [4], this 2DFA has an equivalent *halting* 2DFA with $\leq 4{\cdot}(2{+}s)$ states. This is our M'.

To recap, T_2 iterates over all $O(s^2)$ arrows of K_{2s+1} and then over all $O(s)$ first steps of M in each corresponding segment, finally simulating a $O(s)$-state 2DFA in each iteration. Easily, T_2 needs no more than $O(s^4)$ states and $O(s^2)$ printing transitions, each used at most once. This proves our claim for f_2.

To prime-compute $f_1(x)$ from $x \in \Sigma^*$, a 2DFT T_1 must print a prime encoding $\#z_1 \cdots \#z_m$ of the length of $\langle G(x)\rangle_1$ (making sure no infix $\#z_i$ is split between printing steps). By (2), this length is the product of the primes p_k for which the k-th bit of $\langle G(x)\rangle_2$ is 1. So, m must equal the number of 1s in $T_2(x)$ and each z_i must encode one of the prime powers p_k^1 corresponding to those 1s. Hence, T_1 simulates T_2 and, every time T_2 would print a 1 as its k-th ouput bit, T_1 performs a printing step with output $\#z$, where z is the encoding of p_k^1. Easily, the state diagram of T_1 differs from that of T_2 only in the output strings on the printing transitions. So, the size and print of T_1 are also $O(s^4)$ and $O(s^2)$. □

For two families $\mathcal{T}{=}(T_h)_{h\geq 1}$ of 2DFTs and $\mathcal{F}{=}(f_h)_{h\geq 1}$ of string functions, we say $\mathcal{T}$ (*prime-*) *computes* $\mathcal{F}$ if every T_h (prime-) computes f_h. The members of $\mathcal{T}$ are *laconic* if every T_h performs $\leq p(h)$ printing steps on each input, for some polynomial p. The members of $\mathcal{F}$ are *tight* if $|f_h(x)| \leq p(h){\cdot}|x|$ for some polynomial p, all h, and all x.

For two problem families $\mathcal{L}{=}(\mathfrak{L}_h)_{h\geq 1}$ and $\mathcal{L}'{=}(\mathfrak{L}'_h)_{h\geq 1}$, we say $\mathcal{L}$ *reduces to* $\mathcal{L}'$ *in polynomial size* ($\mathcal{L} \leq_{\mathsf{2D}} \mathcal{L}'$) if every $\mathfrak{L}_h$ reduces to $\mathfrak{L}'_{p(h)}$ for some polynomial p, and the family $\mathcal{F}$ of underlying reductions is computed by a family $\mathcal{T}$ of small 2DFTs; if the problems in $\mathcal{L}'$ are unary and $\mathcal{T}$ prime-computes $\mathcal{F}$, then $\mathcal{L}$ *prime-reduces to* $\mathcal{L}'$ *in polynomial size* ($\mathcal{L} \preceq_{\mathsf{2D}} \mathcal{L}'$); if the 2DFTs in $\mathcal{T}$ are laconic, then $\mathcal{L}$ (*prime-*) *reduces to* $\mathcal{L}'$ *in polynomial size/print* ($\mathcal{L} \leq^{\mathrm{lac}}_{\mathsf{2D}} \mathcal{L}'$ or $\mathcal{L} \preceq^{\mathrm{lac}}_{\mathsf{2D}} \mathcal{L}'$). If every $\mathfrak{L}_h$ homomorphically reduces to $\mathfrak{L}'_{p(h)}$ for some polynomial p, then $\mathcal{L}$ *homomorphically reduces to* $\mathcal{L}'$ ($\mathcal{L} \leq_{\mathrm{h}} \mathcal{L}'$); if the underlying homomorphisms are tight, then $\mathcal{L}$ *reduces to* $\mathcal{L}'$ *under tight homomorphisms* ($\mathcal{L} \leq^{\mathrm{t}}_{\mathrm{h}} \mathcal{L}'$).

As usual, if C is a class of problem families and $\leq$ a type of reductions, then a family $\mathcal{L}$ is C-*complete under* $\leq$ if $\mathcal{L} \in \mathsf{C}$ and every $\mathcal{L}' \in \mathsf{C}$ satisfies $\mathcal{L}' \leq \mathcal{L}$.

Corollary 2. *The following statements are true:*

1. BGAP *is* $\mathsf{2N/poly}$*-complete and* $\mathsf{2O}$*-complete, under polynomial-size/print reductions* ($\leq^{\mathrm{lac}}_{\mathsf{2D}}$).
2. UGAP *is* $\mathsf{2N/unary}$*-complete and* $\mathsf{2O}$*-complete, under polynomial-size/print prime reductions* ($\preceq^{\mathrm{lac}}_{\mathsf{2D}}$).
3. SHORT TWL *is* $\mathsf{2N/poly}$*-complete under tight homomorphic reductions* ($\leq^{\mathrm{t}}_{\mathrm{h}}$).

4 Closures

We now prove that $\mathsf{2D}$ is closed under all of the above reductions. As usual, a class C is *closed* under a type $\leq$ of reductions if $\mathcal{L}_1 \leq \mathcal{L}_2 \;\&\; \mathcal{L}_2 \in \mathsf{C} \Rightarrow \mathcal{L}_1 \in \mathsf{C}$.

Lemma 6. *Suppose* $\mathfrak{L}_2$ *is solved by an* s*-state* 2DFA*. Then the following hold.*

1. *If* $\mathfrak{L}_1 \leq_{\mathrm{m}} \mathfrak{L}_2$ *via a reduction which is computable by an* r*-state* 2DFT *performing* $\leq t$ *printing steps on every input, then* $\mathfrak{L}_1$ *is solved by a* 2DFA *with* $O(rst^2)$ *states.*
2. *If* $\mathfrak{L}_1 \preceq_{\mathrm{m}} \mathfrak{L}_2$ *via a reduction which is prime-computable by an* r*-state* 2DFT*, then* $\mathfrak{L}_1$ *is solved by a* 2DFA *with* $O(rs^2)$ *states.*
3. *If* $\mathfrak{L}_1 \leq_{\mathrm{h}} \mathfrak{L}_2$*, then* $\mathfrak{L}_1$ *is solved by a* 2DFA *with* $2s$ *states.*

Proof. Part 3 is well-known (e.g., see [7, Lemma 2]), so we focus on the first two claims, starting with 1. Suppose $f : \Sigma_1^* \to \Sigma_2^*$ reduces $\mathfrak{L}_1$ to $\mathfrak{L}_2$. Let $T = (S, \Sigma_1, \Sigma_2, ., ., .)$ be a 2DFT that computes f, with $|S| = r$ and $\leq t$ printing steps on every input. Let $M_2 = (S_2, \Sigma_2, ., ., .)$ be a 2DFA that solves $\mathfrak{L}_2$, with $|S_2| = s$. We build a 2DFA $M_1 = (S_1, \Sigma_1, ., ., .)$ for $\mathfrak{L}_1$.

On input $x \in \Sigma_1^*$, M_1 simulates T on x to compute $f(x)$, then simulates M_2 on $f(x)$ and copies its decision. (By the choice of f, this is clearly a correct algorithm for $\mathfrak{L}_1$.) Of course, M_1 cannot store $f(x)$ in between the two simulations. So, it performs them simultaneously: it simulates T on x and, every time T would print a string z (an infix of $f(x)$), M_1 resumes the simulation of M_2 from where it was last interrupted and for as long as it stays within z.

The only problem (a classic, from space complexity) is that T prints $f(x)$ by a one-way head but M_2 reads $f(x)$ by a two-way head. So, whenever the simulation of M_2 falls off the *left* boundary of an infix z, the simulation of T should not continue unaffected to return the next infix after z, but should return again the infix *before* z. The solution (also a classic) is to restart the simulation of T, continue until the desired infix is ready to be output, and then resume the simulation of M_2. This is possible exactly because of the bound t, which implies that $f(x) = z_1 z_2 \cdots z_m$ where $m \leq t$ and z_i is the infix output by T in its i-th printing step. Thus M_1 keeps track of the index i of the infix currently read by the simulation of M_2, and uses it to request z_{i-1} from the next simulation of T.

Easily, M_1 can implement this algorithm with $S_1 := S_2 \times \{\mathrm{L},\mathrm{R}\} \times [t] \times S \times [t]$, where state (q, d, i, p, j) 'means' the simulation of M_2 is ready to resume from

state q on the d-most symbol of z_{i+1}, and the simulation of T (to output z_{i+1}) is in state p and past the printing steps for $z_1, \ldots, z_j$. As claimed, $|S_1| = O(rst^2)$.

Now suppose $\Sigma_2 = \{\mathtt{0}\}$ and T prime-computes f. Then M_1 implements a modification of the above algorithm. Now every string returned by the simulation of T is an infix not of $f(x)$ but of a prime encoding $\mathtt{\#}z_1\mathtt{\#}z_2\mathtt{\#}\cdots\mathtt{\#}z_m$ of $n{:=}|f(x)|$. So, we need to change the way these infixes are treated by the simulation of M_2.

By the analyses of [8,3], it follows that there exist a length bound $l = O(s)$ and a $O(s)$-state RDFA $\tilde{M}_2 = (\tilde{S}_2, \{\mathtt{0}\}, \tilde{\delta}, \tilde{q}_0, \tilde{q}_\mathrm{f})$ such that $\tilde{M}_2$ agrees with M_2 on all lengths $\geq l$, and is in the following 'normal form'. The states $\tilde{S}_2 \setminus \{\tilde{q}_0, \tilde{q}_\mathrm{f}\}$ can be partitioned into a number of cycles $C_1, C_2, \ldots, C_k$. Every rotation on an input y starts on $\vdash$ with a transition into a state $\tilde{q}$ of some C_j, and finishes on $\dashv$ in the state $\tilde{p}$ of the same C_j that is uniquely determined by the entry state $\tilde{q}$, the cycle length $|C_j|$, and the input length $|y|$. From there, the next transition is either off $\dashv$ into $\tilde{q}_\mathrm{f}$ to accept, or back to $\vdash$ and again in $\tilde{p}$ to start a new rotation. Our modified M_1 will use this $\tilde{M}_2$ for checking whether M_2 accepts $f(x)$.

In a first stage, M_1 checks whether $n = |T(x)|$ is one of the lengths $< l$, where $\tilde{M}_2$ and M_2 may disagree; if so, then it halts, accepting iff M_2 accepts $\mathtt{0}^n$; otherwise, it continues to the second stage below. To check whether $n < l$, it iterates over all $\hat{n} \in [l]$, checking for each of them whether $n = \hat{n}$. To check $n = \hat{n}$, it checks whether the prime factorizations of the two numbers contain the same prime powers. This needs $1{+}\hat{m}$ simulations of T on x, for $\hat{m} \leq \log \hat{n}$ the number of prime powers in the factorization of $\hat{n}$: during the 0-th simulation, M_1 checks that every infix $\mathtt{\#}z_i$ of every string output by T encodes some prime power of $\hat{n}$; during the j-th simulation ($1 \leq j \leq \hat{m}$), M_1 checks that the j-th prime power of $\hat{n}$ (under some fixed ordering of prime powers) is encoded by some infix $\mathtt{\#}z_i$ of some string output by T. Overall, this first stage can be implemented on the state set $[l] \times [1 + \log l] \times S$, where state $(\hat{n}, j, q)$ 'means' that the j-th simulation of T for checking $n = \hat{n}$ is currently in state q.

In the second stage, $n \geq l$ is guaranteed, so M_1 can simulate $\tilde{M}_2$ instead of M_2. This is done one rotation at a time. Starting the simulation of a single rotation, M_1 knows the entry state $\tilde{q}$ and cycle C_j to be used; the goal is to compute the state $\tilde{p}$ of C_j when the rotation reaches $\dashv$. Clearly, this reduces to computing the remainder $n \bmod l_j$, where $l_j{:=}|C_j|$ is the cycle length. If n_i is the prime power encoded by the infix $\mathtt{\#}z_i$ of $T(x)$, then $n \bmod l_j$ equals

$$\left(\prod_{i=1}^{m} n_i\right) \bmod l_j = \big(\big(\cdots\big((n_1 \bmod l_j) \cdot n_2\big) \bmod l_j \cdots\big) \cdot n_m\big) \bmod l_j\,. \qquad (3)$$

So, to compute the value ρ of this remainder, M_1 simulates T on x, building ρ as in (3): $\rho \leftarrow 1$ at start, and $\rho \leftarrow (\rho \cdot n_i) \bmod l_j$ for every infix $\mathtt{\#}z_i$ inside a string printed by T. When the simulation of T halts, M_1 finds $\tilde{p}$ in C_j at distance ρ from $\tilde{q}$. From there, if $\tilde{\delta}(\tilde{p}, \dashv) = \tilde{q}_\mathrm{f}$ then $\tilde{M}_2$ accepts, and so does M_1; otherwise, $\tilde{M}_2$ starts a new rotation from state $\tilde{\delta}(\tilde{\delta}(\tilde{p}, \dashv), \vdash)$, and M_1 goes on to simulate it, too. Overall, the second stage can be implemented on the state set $\tilde{S}_2 \times S \times [\,|\tilde{S}_2|\,]$, where state $(\tilde{q}, q, \rho)$ 'means' that the simulation of T for simulating a rotation of $\tilde{M}_2$ from state $\tilde{q}$ is currently in state q and the remainder is currently ρ.

Overall, $S_1{:=}([l] \times [1 + \log l] \times S) \cup (\tilde{S}_2 \times S \times [\,|\tilde{S}_2|\,])$. Hence $|S_1| = O(rs^2)$. $\square$

Corollary 3. 2D *is closed under polynomial-size/print reductions* ($\leq^{\mathrm{lac}}_{\mathsf{2D}}$)*, under polynomial-size prime reductions* ($\preceq_{\mathsf{2D}}$)*, and under homomorphic reductions* ($\leq_{\mathrm{h}}$)*.*

5 Characterizations and Conjectures

Our main theorem is now a direct consequence of [7] and Corollaries 2 and 3.

Theorem 1. *The following statements are equivalent to* $\mathsf{L/poly} \supseteq \mathsf{NL}$*:*

1. $\mathsf{2D} \supseteq \mathsf{2N/poly}$ **3.** $\mathsf{2D} \supseteq \mathsf{2N/unary}$ **5.** $\mathsf{2D} \supseteq \mathsf{2O}$

2. $\mathsf{2D} \ni$ BGAP **4.** $\mathsf{2D} \ni$ UGAP **6.** $\mathsf{2D} \ni$ SHORT TWL

Proof. We have $\mathsf{L/poly} \supseteq \mathsf{NL}$ iff (1), by [7]; (1)⇔(2)⇔(5), by Cor. 2.1 and the closure of 2D under polynomial-size/print reductions; (3)⇔(4)⇔(5) by Cor. 2.2 and the closure of 2D under polynomial-size prime reductions; and (1)⇔(6) by Cor. 2.3 and the closure of 2D under homomorphic reductions. □

The statements of Th. 1 are believed to be false. In particular (statement 6), it is conjectured that no poly(h)-state 2DFA can check liveness on inputs of height h and length $\leq h$. It is thus natural to study shorter inputs. In fact, even inputs of length 2 are interesting: *How large need a* 2DFA *be to solve* COMPACT TWL$_h$*?* Although it can be shown (by Savitch's algorithm) that $2^{O(\log^2 h)}$ states are enough, it is not known whether this can be reduced to poly(h). We conjecture that it cannot. In other words, we conjecture that $\mathsf{L/poly} \not\supseteq \mathsf{NL}$ because already:

Conjecture A. $\mathsf{2D} \not\ni$ COMPACT TWL.

We find this conjecture quite appealing, because of its simple and symmetric setup: just one 2DFA working on just two symbols. Still, this is an *algorithmic* statement (it claims the inexistence of an algorithm), engaging our algorithmic intuitions. These are surely welcome when we need to discover an algorithm, but may be unfavorable when we need to prove that no algorithm exists. It could thus be advantageous to complement this statement with an equivalent one which is purely *combinatorial*. Indeed, such a statement exists: it says that we cannot homomorphically reduce relational to functional match (cf. p. 222).

Conjecture B. REL MATCH $\not\leq_{\mathrm{h}}$ FUN MATCH.

To get a feel of this conjecture, it is useful to note first that RM$_h$ $\leq_{\mathrm{h}}$ FM$_{2^{h^2}}$, and then that this does not imply REL MATCH $\leq_{\mathrm{h}}$ FUN MATCH, because of the super-polynomial blow-up. So, Conjecture B claims that this blow-up cannot be made poly(h) or, more intuitively, that *no systematic way of replacing* h*-tall relations with* poly(h)*-tall functions respects the existence of cycles.*

To prove the equivalence of the two conjectures, we first show that checking for cycles is $\leq_{\mathrm{h}}$-equivalent to checking for cycles through the top left vertex.

Lemma 7. *The following reductions hold:*

1. REL ZERO-MATCH $\leq_{\mathrm{h}}$ REL MATCH *and* REL MATCH $\leq_{\mathrm{h}}$ REL ZERO-MATCH.

2. FUN ZERO-MATCH $\leq_{\mathrm{h}}$ FUN MATCH *and* FUN MATCH $\leq_{\mathrm{h}}$ FUN ZERO-MATCH.

Using this, we then prove that COMPACT TWL and REL MATCH reduce to each other in such a way that small 2DFAs can solve either both or neither, and that REL MATCH is solvable by small 2DFAs iff it $\leq_{\mathrm{h}}$-reduces to FUN MATCH.

Lemma 8. *The following statements hold:*
1. COMPACT TWL $\leq_{\mathsf{2D}}^{\mathrm{lac}}$ REL MATCH *and* REL MATCH $\leq_{\mathrm{h}}$ COMPACT TWL.
2. $\mathsf{2D} \ni$ REL MATCH *iff* REL MATCH $\leq_{\mathrm{h}}$ FUN MATCH.

Combining the two facts, we see that Conjectures A and B are indeed equivalent.

6 Conclusion

We proved several characterizations of $\mathsf{L/poly}$ versus NL in terms of unary, binary, or general 2FAs. Our main theorem complements two recent improvements [5,7] of an old theorem [1], and our approach introduced some of the concepts and tools that had been asked for in [6] for enriching the framework of [9].

It would be interesting to see similar characterizations in terms of unary 2FAs for the uniform variant of the question (L versus NL), or for variants for other bounds for space and advice length (e.g., $\mathsf{LL/polylog}$ versus NLL [7]). Another interesting direction is to elaborate further on the comparison between 2FA computations on short and on unary inputs; for example, using the ideas of this article, one can show that for every $\mathcal{L} \in \mathsf{2N/poly}$ there is $\mathcal{L}' \in \mathsf{2N/unary}$ such that $\mathcal{L} \preceq_{\mathsf{2D}}^{\mathrm{lac}} \mathcal{L}'$. Finally, further work is needed to fully understand the capabilities and limitations of the reductions introduced in this article.

References

1. Berman, P., Lingas, A.: On complexity of regular languages in terms of finite automata. Report 304, Institute of Computer Science, Polish Academy of Sciences, Warsaw (1977)
2. Geffert, V., Guillon, B., Pighizzini, G.: Two-Way Automata Making Choices Only at the Endmarkers. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 264–276. Springer, Heidelberg (2012)
3. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic unary automata into simpler automata. Theoretical Computer Science 295, 189–203 (2003)
4. Geffert, V., Mereghetti, C., Pighizzini, G.: Complementing two-way finite automata. Information and Computation 205(8), 1173–1187 (2007)
5. Geffert, V., Pighizzini, G.: Two-way unary automata versus logarithmic space. Information and Computation 209(7), 1016–1025 (2011)
6. Kapoutsis, C.A.: Size Complexity of Two-Way Finite Automata. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 47–66. Springer, Heidelberg (2009)
7. Kapoutsis, C.A.: Two-Way Automata versus Logarithmic Space. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 359–372. Springer, Heidelberg (2011)
8. Kunc, M., Okhotin, A.: Describing Periodicity in Two-Way Deterministic Finite Automata Using Transformation Semigroups. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 324–336. Springer, Heidelberg (2011)
9. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Proceedings of STOC, pp. 275–286 (1978)

Cutting through Regular Post Embedding Problems*

Prateek Karandikar[1] and Philippe Schnoebelen[2]

[1] Chennai Mathematical Institute
[2] LSV, ENS Cachan, CNRS

Abstract. The Regular Post Embedding Problem extended with partial (co)directness is shown decidable. This extends to universal and/or counting versions. It is also shown that combining directness and codirectness in Post Embedding problems leads to undecidability.

1 Introduction

The Regular Post Embedding Problem. (PEP for short, named by analogy with Post's Correspondence Problem) is the problem of deciding, given two morphisms on words $u, v : \Sigma^* \to \Gamma^*$ and a regular language $R \in \mathsf{Reg}(\Sigma)$, whether there is $\sigma \in R$ such that $u(\sigma)$ is a (scattered) subword of $v(\sigma)$. The *subword* ordering, also called *embedding*, is denoted "$\sqsubseteq$": $u(\sigma) \sqsubseteq v(\sigma) \stackrel{\text{def}}{\Leftrightarrow} u(\sigma)$ can be obtained by erasing some letters from $v(\sigma)$, possibly all of them, possibly none. Equivalently, PEP is the question whether a rational relation, or a transduction, $T \subseteq \Gamma^* \times \Gamma^*$ intersects non-vacuously the subword relation, hence is a special case of the intersection problem for two rational relations.

This problem is new and quite remarkable: it is decidable [2] but surprisingly hard since it is not primitive-recursive and not even multiply-recursive. In fact, it is at level $\mathcal{F}_{\omega^\omega}$ (and not below) in the Fast-Growing Hierarchy [8,12].

A variant problem, $\mathsf{PEP}_{\text{dir}}$, asks for the existence of *direct* solutions, i.e., solutions $\sigma \in R$ such that $u(\tau) \sqsubseteq v(\tau)$ for every prefix τ of σ. The two problems are inter-reducible [4], hence have the same complexity: decidability of PEP entails decidability of $\mathsf{PEP}_{\text{dir}}$, while hardness of $\mathsf{PEP}_{\text{dir}}$ entails hardness for PEP.

Our contribution. We introduce $\mathsf{PEP}_{\text{dir}}^{\text{partial}}$, or "PEP with *partial* directness", a new problem that generalizes both PEP and $\mathsf{PEP}_{\text{dir}}$, and prove its decidability. The proof combines two ideas. Firstly, by Higman's Lemma, a long solution must eventually contain "*comparable*" so-called cutting points, from which one deduces that the solution is not minimal (or unique, or ...). Secondly, the above notion of "*eventually*", that comes from Higman's Lemma, can be turned into an effective upper bound thanks to a Length Function Theorem. The cutting technique described above was first used in [7] for reducing $\exists^\infty$PEP to PEP. In this paper we use it to obtain a decidability proof for $\mathsf{PEP}_{\text{dir}}^{\text{partial}}$ that is not only more general but also more direct than the earlier proofs for PEP or $\mathsf{PEP}_{\text{dir}}$. It also immediately provides an $\mathcal{F}_{\omega^\omega}$ complexity upper bound. We also

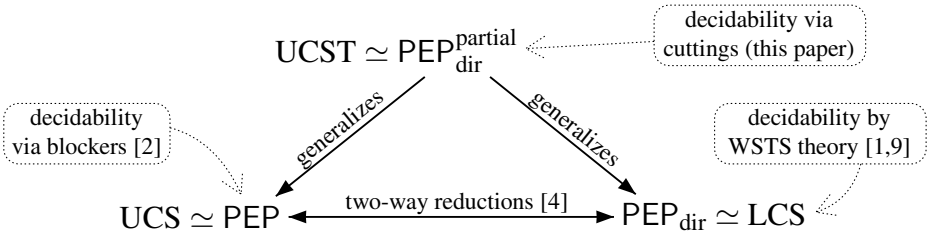
* Supported by ARCUS 2008–11 Île de France-Inde and Grant ANR-11-BS02-001. The first author was partially funded by Tata Consultancy Services.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 229–240, 2012.

show the decidability of universal and/or counting versions of the extended $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ problem, and explain how our attempts at further generalisation, most notably by considering the combination of directness and codirectness in a same instance, lead to undecidability.

Applications to channel machines. Beyond the tantalizing decidability questions, our interest in PEP and its variants comes from their close connection with fifo channel machines [11], a family of computational models that are a central tool in several areas of program and system verification (see [5] and the references therein). Here, PEP and its variants provide abstract versions of verification problems for channel machines [4], bringing greater clarity and versatility in both decidability and undecidability (more generally, hardness) proofs.

Beyond providing a uniform and simpler proof for the decidability of PEP and $\mathsf{PEP}_{\text{dir}}$, our motivation for considering $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is that it allows solving the decidability of UCST, i.e., unidirectional channel systems (with one reliable and one lossy channel) *extended with the possibility of testing the contents of channels* [10]. We recall that PEP was introduced for UCS, unidirectional channel systems where tests on channels are not supported [4,3], and that $\mathsf{PEP}_{\text{dir}}$ corresponds to LCS, i.e., lossy channel systems, for which verification is decidable using techniques from WSTS theory [1,9,5]. The following figure illustrates the resulting situation.



Outline of the paper. Section 2 recalls basic notations and definitions. In particular, it explains the Length Function Theorem for Higman's Lemma, and lists basic results where the subword relation interacts with concatenations and factorization. Section 3 contains our main result, a direct decidability proof for $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$, a problem subsuming both PEP and $\mathsf{PEP}_{\text{dir}}$. Section 4 builds on this result and shows the decidability of counting problems on $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$. Section 5 further shows the decidability of universal variants of these questions. Section 6 contains our undecidability results for extensions of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$. Proofs omitted for lack of space can be found in the full version of this paper, available at `arxiv.org/abs/1109.1691`.

2 Basic Notation and Definitions

Words. Concatenation of words is denoted multiplicatively, with ε denoting the empty word. If s is a prefix of a word t, $s^{-1}t$ denotes the unique word s' such that $t = ss'$, and $s^{-1}t$ is not defined if s is not a prefix of t. Similarly, when s is a suffix of t, ts^{-1} is t with the s suffix removed. For a word $x = \mathtt{a}_0 \dots \mathtt{a}_{n-1}$, $\tilde{x} \stackrel{\text{def}}{=} \mathtt{a}_{n-1} \dots \mathtt{a}_0$ is the mirrored word.

The mirror of a language R is $\tilde{R} \stackrel{\text{def}}{=} \{\tilde{x} \mid x \in R\}$. We write $s \sqsubseteq t$ when s is a subword (subsequence) of t.

Lemma 2.1 (Subwords and concatenation). *For all words y,z,s,t:*
1. If $yz \sqsubseteq st$ then $y \sqsubseteq s$ or $z \sqsubseteq t$.
2. If $yz \sqsubseteq st$ and $z \sqsubseteq t$ and x is the longest suffix of y such that $xz \sqsubseteq t$, then $yx^{-1} \sqsubseteq s$.
3. If $yz \sqsubseteq st$ and $z \not\sqsubseteq t$ and x is the shortest prefix of z such that $x^{-1}z \sqsubseteq t$, then $yx \sqsubseteq s$.
4. If $yz \sqsubseteq st$ and $z \sqsubseteq t$ and x is the longest prefix of t such that $z \sqsubseteq x^{-1}t$, then $y \sqsubseteq sx$.
5. If $yz \sqsubseteq st$ and $z \not\sqsubseteq t$ and x is the shortest suffix of s such that $z \sqsubseteq xt$, then $y \sqsubseteq sx^{-1}$.
6. If $sx \sqsubseteq yt$ and $t \sqsubseteq s$, then $sx^k \sqsubseteq y^k t$ for all $k \geq 1$.
7. If $xs \sqsubseteq ty$ and $t \sqsubseteq s$, then $x^k s \sqsubseteq ty^k$ for all $k \geq 1$.

With a language R one associates a congruence (wrt concatenation) given by $s \sim_R t \stackrel{\text{def}}{\Leftrightarrow} \forall x,y(xsy \in R \Leftrightarrow xty \in R)$ and called the syntactic congruence (also, the syntactic monoid). This congruence has finite index if (and only if) R is regular. For regular R, let n_R denote this index: $n_R \leq m^m$ when R is recognized by a m-state complete DFA.

Higman's Lemma. It is well-known that for words over a finite alphabet, $\sqsubseteq$ is a well-quasi-ordering, that is, any infinite sequence of words $x_1, x_2, x_3, \ldots$ contains an infinite increasing subsequence $x_{i_1} \sqsubseteq x_{i_2} \sqsubseteq x_{i_3} \sqsubseteq \cdots$. This result is called Higman's Lemma.

For $n \in \mathbb{N}$, we say that a sequence (finite or infinite) of words is n-good if it has an increasing subsequence of length n. It is n-bad otherwise. Higman's Lemma tells us that every infinite sequence is n-good for every n. Hence every n-bad sequence is finite.

It is often said that Higman's Lemma is "non-effective" since it does not give any explicit information on the maximal length of bad sequences. Consequently, when one uses Higman's Lemma to prove that an algorithm terminates, no meaningful upper-bound on the algorithm's running time is derived from the proof. However, complexity upper-bound can be derived if the complexity of the sequences (or more precisely of the process that generates bad sequences) is taken into account. The interested reader can consult [12] for more details. Here we only need the simplest version of these results, i.e., the statement that the maximal length of bad sequences is computable.

A sequence of words $x_1, \ldots, x_l$ is *k-controlled* ($k \in \mathbb{N}$) if $|x_i| \leq ik$ for all $i = 1, \ldots, l$.

Length Function Theorem. *There exists a computable function $H : \mathbb{N}^3 \to \mathbb{N}$ such that any n-bad k-controlled sequences of words in Γ^* has length at most $H(n,k,|\Gamma|)$. Furthermore, H is monotonic in all three arguments.*

Thus, a sequence with more than $H(n,k,|\Gamma|)$ words is n-good or is not k-controlled. We refer to [12] for the complexity of H. Here it is enough to know that H is computable.

3 Deciding $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$, or PEP with Partial Directness

We introduce $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$, a problem generalizing both PEP and $\mathsf{PEP}_{\text{dir}}$, and show its decidability. This is proved by showing that if a $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ instance has a solution, then it has a solution whose length is bounded by a computable function of the input. This proof is simpler and more direct than the proof (for PEP only) based on blockers [2].

Definition 3.1. $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ *is the problem of deciding, given morphisms* $u, v : \Sigma^* \to \Gamma^*$ *and regular languages* $R, R' \in \mathsf{Reg}(\Sigma)$*, whether there is* $\sigma \in R$ *such that* $u(\sigma) \sqsubseteq v(\sigma)$ *and* $u(\tau) \sqsubseteq v(\tau)$ *for all prefixes* τ *of* σ *belonging to* R' *(in which case* σ *is called* a solution*).* $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ *is the variant problem of deciding whether there is* $\sigma \in R$ *such that* $u(\sigma) \sqsubseteq v(\sigma)$ *and* $u(\tau) \sqsubseteq v(\tau)$ *for all suffixes of* τ *of* σ *that belong to* R'*.*

Both PEP and $\mathsf{PEP}_{\text{dir}}$ are special cases of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$, obtained by taking $R' = \varnothing$ and $R' = \Sigma^*$ respectively. Obviously $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ are two equivalent presentations, modulo mirroring, of a same problem. Given a $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ (or $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$) instance, we let $K_u \stackrel{\text{def}}{=} \max_{a\in\Sigma} |u(a)|$ denote the *expansion factor* of u and say that $\sigma \in \Sigma^*$ is *long* if $|\sigma| > 2H(n_R n_{R'} + 1, K_u, |\Gamma|)$, otherwise it is *short* (recall that $H(n,k,|\Gamma|)$ was defined with the Length Function Theorem). In this section we prove:

Theorem 3.2. *A* $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ *or* $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ *instance has a solution if, and only if, it has a short solution. This entails that* $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ *and* $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ *are decidable.*

Decidability is an obvious consequence since the maximal length for short solutions is computable, and since it is easy to check whether a candidate σ is a solution. Furthermore, one derives an upper bound on the complexity of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ since the Length Function H is bounded in $\mathcal{F}_{\omega^\omega}$ [12].

For the proof of Theorem 3.2, we find it easier to reason on the codirect version. Pick an arbitrary $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance $(\Sigma, \Gamma, u, v, R, R')$ and a solution σ. Write $N = |\sigma|$ for its length, $\sigma[0,i)$ and $\sigma[i,N)$ for, respectively, its prefix of length i and its suffix of length $N-i$. Two indices $i, j \in [0,N]$ are *congruent* if $\sigma[i,N) \sim_R \sigma[j,N)$ and $\sigma[i,N) \sim_{R'} \sigma[j,N)$. When σ is fixed, as in the rest of this section, we use shorthand notations like $u_{0,i}$ and $v_{i,j}$ to denote the images, here $u(\sigma[0,i))$ and $v(\sigma[i,j))$, of factors of σ.

We prove two "cutting lemmas" giving sufficient conditions for "cutting" a solution $\sigma = \sigma[0,N)$ along certain indices $a < b$, yielding a shorter solution $\sigma' = \sigma[0,a)\sigma[b,N)$. Here the following notation is useful. We associate, with every suffix τ of σ', a corresponding suffix, denoted $S(\tau)$, of σ: if τ is a suffix of $\sigma[b,N)$, then $S(\tau) \stackrel{\text{def}}{=} \tau$, otherwise, $\tau = \sigma[i,a)\sigma[b,N)$ for some $i < a$ and we let $S(\tau) \stackrel{\text{def}}{=} \sigma[i,N)$. In particular $S(\sigma') = \sigma$.

An index $i \in [0,N]$ is said to be *blue* if $u_{i,N} \sqsubseteq v_{i,N}$, it is *red* otherwise. In particular, N is blue trivially, 0 is blue since σ is a solution, and i is blue whenever $\sigma[i,N) \in R'$. If i is a blue index, let $l_i \in \Gamma^*$ be the longest suffix of $u_{0,i}$ such that $l_i u_{i,N} \sqsubseteq v_{i,N}$ and call it the *left margin* at i.

Lemma 3.3 (Cutting lemma for blue indices). *Let* $a < b$ *be two congruent and blue indices. If* $l_a \sqsubseteq l_b$*, then* $\sigma' = \sigma[0,a)\sigma[b,N)$ *is a solution (shorter than* σ*).*

Proof. Clearly $\sigma' \in R$ since $\sigma \in R$ and a and b are congruent. Also, for all suffixes τ of σ', $S(\tau) \in R'$ iff $\tau \in R'$.

We claim that, for any suffix τ of σ', if $u(S(\tau)) \sqsubseteq v(S(\tau))$ then $u(\tau) \sqsubseteq v(\tau)$. This is obvious when $\tau = S(\tau)$, so we assume $\tau \neq S(\tau)$, i.e., $\tau = \sigma[i,a)\sigma[b,N)$ and $S(\tau) = \sigma[i,N)$ for some $i < a$. Assume $u(S(\tau)) \sqsubseteq v(S(\tau))$, i.e., $u_{i,N} \sqsubseteq v_{i,N}$. Now at least one of $u_{i,a}$ and l_a is a suffix of the other, which gives two cases. If $u_{i,a}$ is a suffix of l_a, then

$$u(\tau) = u_{i,a}u_{b,N} \sqsubseteq l_a u_{b,N} \sqsubseteq l_b u_{b,N} \sqsubseteq v_{b,N} \sqsubseteq v(\tau)\,.$$

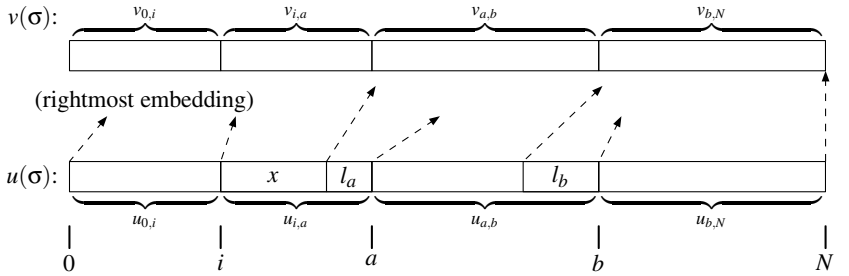


Fig. 1. Schematics for Lemma 3.3, with $l_a \sqsubseteq l_b$

Otherwise, $u_{i,a} = xl_a$ for some x (see Fig. 1). Then $u_{i,N} \sqsubseteq v_{i,N}$ rewrites as $u_{i,a}u_{a,N} = xl_au_{a,N} \sqsubseteq v_{i,a}v_{a,N}$. Now, and since l_a is the longest suffix for which $l_au_{a,N} \sqsubseteq v_{a,N}$, Lemma 2.1.2 entails $x \sqsubseteq v_{i,a}$. Combining with $l_a \sqsubseteq l_b$ (assumption of the Lemma) gives:

$$u(\tau) = u_{i,a}u_{b,N} = xl_au_{b,N} \sqsubseteq v_{i,a}l_bu_{b,N} \sqsubseteq v_{i,a}v_{b,N} = v(\tau)\,.$$

This shows that σ' is a solution (which completes the proof) since we can infer $u(\tau) \sqsubseteq v(\tau)$ for any suffix $\tau \in R'$ (or for $\tau = \sigma'$) from the corresponding $u(S(\tau)) \sqsubseteq v(S(\tau))$. □

If i is a red index, let $r_i \in \Gamma^*$ be the shortest prefix of $u_{i,N}$ such that $r_i^{-1}u_{i,N} \sqsubseteq v_{i,N}$ (equivalently $u_{i,N} \sqsubseteq r_iv_{i,N}$) and call it the *right margin* at i.

Lemma 3.4 (Cutting lemma for red indices). *Let $a < b$ be two congruent and red indices. If $r_b \sqsubseteq r_a$, then $\sigma' = \sigma[0,a)\sigma[b,N)$ is a solution (shorter than σ).*

Proof. Write $u_{b,N}$ under the form r_bx so that $x \sqsubseteq v_{b,N}$. We proceed as for Lemma 3.3 and show that $u(S(\tau)) \sqsubseteq v(S(\tau))$ implies $u(\tau) \sqsubseteq v(\tau)$ for all suffixes τ of σ'. Assume $u(S(\tau)) \sqsubseteq v(S(\tau))$ for some τ. The only interesting case is when $\tau \neq S(\tau)$ and $\tau = \sigma[i,a)\sigma[b,N)$ for some $i < a$ (see Fig. 2).

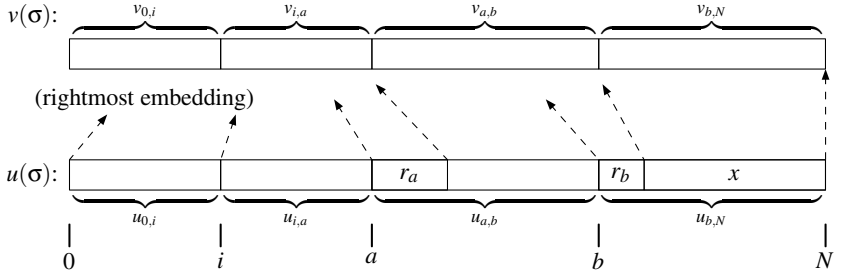


Fig. 2. Schematics for Lemma 3.4, with $r_b \sqsubseteq r_a$

From $u_{i,N} = u_{i,a}u_{a,N} \sqsubseteq v_{i,a}v_{a,N} = v_{i,N}$, i.e., $u(S(\tau)) \sqsubseteq v(S(\tau))$, and $u_{a,N} \not\sqsubseteq v_{a,N}$ (since a is a red index), the definition of r_a entails $u_{i,a}r_a \sqsubseteq v_{i,a}$ (Lemma 2.1.3). Then

$$u(\tau) = u_{i,a}u_{b,N} = u_{i,a}r_bx \sqsubseteq u_{i,a}r_av_{b,N} \sqsubseteq v_{i,a}v_{b,N} = v(\tau)\,. \qquad \square$$

We now conclude the proof of Theorem 3.2. Let $g_1 < g_2 < \cdots < g_{N_1}$ be the blue indices in σ, let $b_1 < b_2 < \cdots < b_{N_2}$ be the red indices, and look at the corresponding sequences $(l_{g_i})_{i=1,\ldots,N_1}$ of left margins and $(r_{b_i})_{i=1,\ldots,N_2}$ of right margins.

Lemma 3.5. *$|l_{g_i}| \leq (i-1) \times K_u$ for all $i = 1,\ldots,N_1$, and $|r_{b_i}| \leq (N_2 - i + 1) \times K_u$ for all $i = 1,\ldots,N_2$. In other words, the sequence on left margins and the* reversed *sequence of right margins are K_u-controlled.*

Now, let $N_c \stackrel{\text{def}}{=} n_R n_{R'} + 1$ and $L \stackrel{\text{def}}{=} H(N_c, K_u, |\Gamma|)$ and assume $N > 2L$. Since $N_1 + N_2 = N + 1$, either σ has at least $L+1$ blue indices and, by definition of L and H, there exist N_c blue indices $a_1 < a_2 < \cdots < a_{N_c}$ with $l_{a_1} \sqsubseteq l_{a_2} \sqsubseteq \cdots \sqsubseteq l_{a_{N_c}}$, or σ has at least $L+1$ red indices and there exist N_c red indices $a'_1 < a'_2 < \cdots < a'_{N_c}$ with $r_{a'_{N_c}} \sqsubseteq \cdots \sqsubseteq r_{a'_2} \sqsubseteq r_{a'_1}$ (since it is the reversed sequence of right margins that is controlled). Out of $N_c = 1 + n_R n_{R'}$ indices, two must be congruent, fulfilling the assumptions of Lemma 3.3 or Lemma 3.4. Therefore σ is not the shortest solution, proving Theorem 3.2.

4 Counting the Number of Solutions

We consider two counting questions: $\exists^{\infty}\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is the question whether a $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ instance has infinitely many solutions (a decision problem), while $\#\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is the problem of computing the number of solutions of the instance (a number in $\mathbb{N} \cup \{\infty\}$). For technical convenience, we often deal with the (equivalent) codirected versions, $\exists^{\infty}\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and $\#\mathsf{PEP}^{\text{partial}}_{\text{codir}}$.

For an instance $(\Sigma, \Gamma, u, v, R, R')$, we let $K_v \stackrel{\text{def}}{=} \max_{a\in\Sigma} |v(a)|$ and define

$$L \stackrel{\text{def}}{=} H(n_R n_{R'} + 1, K_v, |\Gamma|)\,, \qquad L' \stackrel{\text{def}}{=} H\Big(\big[n_R(2L+1)\big]^{n_R(2L+1)} n_{R'} + 1, K_u, |\Gamma|\Big)\,.$$

We say that a solution $\sigma \in \Sigma^*$ is *long* if $|\sigma| > 2L$ and *very long* if $|\sigma| > 2L'$ (note that "long" is slightly different from "not short" from Section 3). In this section we prove:

Theorem 4.1. *For a $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ or $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance, the following are equivalent:*
(a). It has infinitely many solutions.
(b). It has a long solution.
(c). It has a solution that is long but not very long.

From this, it will be easy to count the number of solutions:

Corollary 4.2. *$\exists^{\infty}\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and $\exists^{\infty}\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ are decidable, $\#\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and $\#\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ are computable.*

Proof. Decidability for the decision problems is clear since L and L' are computable.

For actually counting the solutions, we check whether the number of solutions is finite or not using the decision problems. If infinite, we are done. If finite, we first compute an upper bound on the length of the longest solution. For this we build $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ (resp. $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$) instances where R is replaced by $R \smallsetminus \Sigma^{\leq M}$ (which is regular when R is) for increasing values of $M \in \mathbb{N}$. When eventually M is large enough, the instance is negative and this can be detected (by Theorem 3.2). Once we know that there are no solutions longer than M, counting solutions is done by finite enumeration. □

We now prove Theorem 4.1. First observe that if the instance has a long solution, it has a solution with R replaced by $R \cap \Sigma^{>2L}$. This language has a DFA with $n_R(2L+1)$ states, thus the associated congruence has index at most $(n_R(2L+1))^{n_R(2L+1)}$. From Theorem 3.2, the instance has a solution which is long but not very long. Hence (b) and (c) are equivalent.

It remains to show (b) implies (a) since obviously (a) implies (b). For this we fix an arbitrary $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance $(\Sigma, \Gamma, u, v, R, R')$ and consider a solution σ, of length N. We develop two so-called "iteration lemmas" that are similar to the cutting lemmas from Section 3, with the difference that they expand σ instead of reducing it.

As before, an index $i \in [0,N]$ is said to be *blue* if $u_{i,N} \sqsubseteq v_{i,N}$, and *red* otherwise. With blue and red indices we associate words analogous to the l_i's and r_i's from Section 3, however now they are factors of $v(\sigma)$, not $u(\sigma)$ (hence the different definition for L). The terms "left margin" and "right margin" will be reused here for these factors.

We start with blue indices. For a blue index $i \in [0,N]$, let s_i be the longest prefix of $v_{i,N}$ such that $u_{i,N} \sqsubseteq s_i^{-1} v_{i,N}$ (equivalently $s_i u_{i,N} \sqsubseteq v_{i,N}$) and call it the *right margin* at i.

Lemma 4.3. *Suppose $a < b$ are two blue indices with $s_b \sqsubseteq s_a$. Then for all $k \geq 1$, $s_a (u_{a,b})^k \sqsubseteq (v_{a,b})^k s_b$.*

Proof. $s_a u_{a,N} \sqsubseteq v_{a,N}$ expands as $(s_a u_{a,b}) u_{b,N} \sqsubseteq v_{a,b} v_{b,N}$. Since $u_{b,N} \sqsubseteq v_{b,N}$, Lemma 2.1.4 yields $s_a u_{a,b} \sqsubseteq v_{a,b} s_b$. One concludes with Lemma 2.1.6, using $s_b \sqsubseteq s_a$. □

Lemma 4.4 (Iteration lemma for blue indices). *Let $a < b$ be two congruent and blue indices. If $s_b \sqsubseteq s_a$, then for every $k \geq 1$, $\sigma' = \sigma[0,a).\sigma[a,b)^k.\sigma[b,N)$ is a solution.*

Now to red indices. For a red index $i \in [0,N]$, let t_i be the shortest suffix of $v_{0,i}$ such that $u_{i,N} \sqsubseteq t_i v_{i,N}$. This is called the *left margin* at i. Thus, for a blue j such that $j < i$, $u_{j,N} \sqsubseteq v_{j,N}$ implies $u_{j,i} t_i \sqsubseteq v_{j,i}$ by Lemma 2.1.5.

Lemma 4.5 (Iteration lemma for red indices). *Let $a < b$ be two congruent and red indices. If $t_a \sqsubseteq t_b$, then for every $k \geq 1$, $\sigma' = \sigma[0,a).\sigma[a,b)^k.\sigma[b,N)$ is a solution.*

We now conclude the proof of Theorem 4.1. We first prove that the $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance has infinitely many solutions iff it has a long solution. Obviously, only the right-to-left implication has to be proven.

Suppose there are N_1 blue indices in σ, say $g_1 < g_2 < \cdots < g_{N_1}$; and N_2 red indices, say $b_1 < b_2 < \cdots < b_{N_2}$.

Lemma 4.6. *$|s_{g_i}| \leq (N_1 - i + 1) \times K_v$ for all $i = 1, \ldots, N_1$, and $|t_{b_i}| \leq (i-1) \times K_v$ for all $i = 1, \ldots, N_2$. That is, the* reversed *sequence of right margins and the sequence of left margins are K_v-controlled.*

Assume that σ is a long solution of length $N \geq 2L+1$. At least $L+1$ indices among $[0,N]$ are blue, or at least $L+1$ are red. We apply one of the two above claims, and from either $s_{g_{N_1}}, \ldots, s_{g_1}$ (if $N_1 \geq L+1$) or $t_{b_1}, \ldots, t_{b_{N_2}}$ (if $N_2 \geq L+1$) we get an increasing subsequence of length $n_R n_{R'} + 1$. Among these there must be two congruent indices. Then we get infinitely many solutions by Lemma 4.4 or Lemma 4.5.

5 Universal Variants of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$

We consider universal variants of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ (or rather $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ for the sake of uniformity). Formally, given instances $(\Sigma, \Gamma, u, v, R, R')$ as usual, $\forall\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is the question whether *every* $\sigma \in R$ is a solution, i.e., satisfies both $u(\sigma) \sqsubseteq v(\sigma)$ and $u(\tau) \sqsubseteq v(\tau)$ for all suffixes τ that belong to R'. Similarly, $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is the question whether "almost all", i.e., *all but finitely many*, σ in R are solutions, and $\#\neg\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is the associated counting problem that asks how many $\sigma \in R$ are not solutions.

The special cases $\forall\mathsf{PEP}$ and $\forall^\infty\mathsf{PEP}$ (where $R' = \varnothing$) have been shown decidable in [7] where it appears that, at least for Post Embedding, universal questions are simpler than existential ones. We now observe that $\forall\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ are easy to solve too: partial codirectness constraints can be eliminated since universal quantifications commute with conjunctions (and since the codirectness constraint is universal itself).

Lemma 5.1. *$\forall\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ many-one reduce to $\forall^\infty\mathsf{PEP}$.*

Corollary 5.2. *$\forall\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ are decidable, $\#\neg\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is computable.*

We now prove Lemma 5.1. First, $\forall\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ easily reduces to $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$: add an extra letter $\mathtt{z}$ to Σ with $u(\mathtt{z}) = v(\mathtt{z}) = \varepsilon$ and replace R and R' with $R.\mathtt{z}^*$ and $R'.\mathtt{z}^*$. Hence the second half of the lemma entails its first half by transitivity of reductions.

For reducing $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$, it is easier to start with the negation of our question:

$$\exists^\infty\sigma \in R : \big(u(\sigma) \not\sqsubseteq v(\sigma) \text{ or } \sigma \text{ has a suffix } \tau \text{ in } R' \text{ with } u(\tau) \not\sqsubseteq v(\tau)\big)\,. \qquad (*)$$

Call $\sigma \in R$ a *type 1 witness* if $u(\sigma) \not\sqsubseteq v(\sigma)$, and a *type 2 witness* if it has a suffix $\tau \in R'$ with $u(\tau) \not\sqsubseteq v(\tau)$. Statement $(*)$ holds if, and only if, there are infinitely many type 1 witnesses or infinitely many type 2 witnesses. The existence of infinitely many type 1 witnesses (call that "case 1") is the negation of a $\forall^\infty\mathsf{PEP}$ question. Now suppose that there are infinitely many type 2 witnesses, say $\sigma_1, \sigma_2, \ldots$ For each i, pick a suffix τ_i of σ_i such that $\tau_i \in R'$ and $u(\tau_i) \not\sqsubseteq v(\tau_i)$. The set $\{\tau_i \mid i = 1, 2, \ldots\}$ of these suffixes can be finite or infinite. If it is infinite ("case 2a"), then

$$u(\tau) \not\sqsubseteq v(\tau) \text{ for infinitely many } \tau \in (\overrightarrow{R} \cap R')\,, \qquad (**)$$

where $\overrightarrow{R}$ is short for $\overrightarrow{\geq 0 R}$ and for $k \in \mathbb{N}$, $\overrightarrow{\geq k R} \stackrel{\text{def}}{=} \{y \mid \exists x : (|x| \geq k \text{ and } xy \in R)\}$ is the set of the suffixes of words from R one obtains by removing at least k letters. Observe that, conversely, $(**)$ implies the existence of infinitely many type 2 witnesses (for a proof, pick $\tau_1 \in \overrightarrow{R} \cap R'$ satisfying the above, choose $\sigma_1 \in R$ of which τ_1 is a suffix. Then choose τ_2 such that $|\tau_2| > |\sigma_1|$, and proceed similarly).

On the other hand, if $\{\tau_i \mid i = 1, 2, \ldots\}$ is finite ("case 2b"), then there is a $\tau \in R'$ such that $u(\tau) \not\sqsubseteq v(\tau)$ and $\sigma'\tau \in R$ for infinitely many σ'. By a standard pumping argument, the second point is equivalent to the existence of some such σ' with also $|\sigma'| > k_R$, where k_R is the size of a NFA for R (taking $k_R = n_R$ also works). Write now $\hat{R}$ for $\overrightarrow{>k_R R}$: if $\{\tau_i \mid i = 1, 2, \ldots\}$ is finite, then $u(\tau) \not\sqsubseteq v(\tau)$ for some τ in $(R' \cap \hat{R})$, and conversely this implies the existence of infinitely many type 2 witnesses.

To summarize, and since $\overrightarrow{R}$ and $\hat{R}$ are regular and effectively computable from R, we have just reduced $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ to the following conjunction

$$\underbrace{\forall^\infty \sigma \in R : u(\sigma) \sqsubseteq v(\sigma)}_{\text{not case 1}} \bigwedge \underbrace{\forall^\infty \tau \in (\overrightarrow{R} \cap R') : u(\tau) \sqsubseteq v(\tau)}_{\text{not case 2a}} \bigwedge \underbrace{\forall \tau \in (\hat{R} \cap R') : u(\tau) \sqsubseteq v(\tau)}_{\text{not case 2b}}.$$

This is now reduced to a single $\forall^\infty\mathsf{PEP}$ instance by rewriting the $\forall\mathsf{PEP}$ into a $\forall^\infty\mathsf{PEP}$ (as said in the beginning of this proof) and relying on distributivity:

$$\bigwedge_{i=1}^{n} \left[\forall^\infty x \in X_i : \dots \text{ some property } \dots\right] \equiv \forall^\infty x \in \bigcup_{i=1}^{n} X_i : \dots \text{ same } \dots .$$

6 Undecidability for $\mathsf{PEP}_{\text{co\&dir}}$ and Other Extensions

The decidability of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is a non-trivial generalization of previous results for PEP. It is a natural question whether one can further generalize the idea of partial directness and maintain decidability. In this section we describe two attempts that lead to undecidability, even though they remain inside the regular PEP framework.[1]

Allowing non-regular R'. One direction for extending $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is to allow *more expressive R' sets* for partial (co)directness. Let $\mathsf{PEP}^{\text{partial[DCFL]}}_{\text{codir}}$ and $\mathsf{PEP}^{\text{partial[Pres]}}_{\text{codir}}$ be like $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ except that R' can be any deterministic context-free $R' \in \mathsf{DCFL}(\Sigma)$ (respectively, any Presburger-definable $R' \in \mathsf{Pres}(\Sigma)$, i.e., a language whose Parikh image is a Presburger, or semilinear, subset of $\mathbb{N}^{|\Sigma|}$). Note that $R \in \mathsf{Reg}(\Sigma)$ is still required.

Theorem 6.1 (Undecidability). $\mathsf{PEP}^{\text{partial[DCFL]}}_{\text{codir}}$ *and* $\mathsf{PEP}^{\text{partial[Pres]}}_{\text{codir}}$ *are Σ^0_1-complete.*

Since both problems clearly are in Σ^0_1, one only has to prove hardness by reduction, e.g., from PCP, Post's Correspondence Problem. Let (Σ, Γ, u, v) be a PCP instance (where the question is whether there exists $x \in \Sigma^+$ such that $u(x) = v(x)$). Extend Σ and Γ with new symbols: $\Sigma' \stackrel{\text{def}}{=} \Sigma \cup \{1,2\}$ and $\Gamma' \stackrel{\text{def}}{=} \Gamma \cup \{\#\}$. Now define $u', v' : \Sigma'^* \to \Gamma'^*$ by extending u, v on the new symbols with $u'(1) = v'(2) = \varepsilon$ and $u'(2) = v'(1) = \#$. Define now $R = 12\Sigma^+$ and $R' = \{\tau 2 \tau' \mid \tau, \tau' \in \Sigma^* \text{ and } |u(\tau\tau')| \neq |v(\tau\tau')|\}$. Note that R' is deterministic context-free and Presburger-definable.

Lemma 6.2. *The* PCP *instance (Σ, Γ, u, v) has a solution if and only if the* $\mathsf{PEP}^{\text{partial[Pres]}}_{\text{codir}}$ *and* $\mathsf{PEP}^{\text{partial[DCFL]}}_{\text{codir}}$ *instance $(\Sigma', \Gamma', u', v', R, R')$ has a solution.*

Combining directness and codirectness. Another direction is to allow *combining* directness and codirectness constraints. Formally, $\mathsf{PEP}_{\text{co\&dir}}$ is the problem of deciding, given Σ, Γ, u, v, and $R \in \mathsf{Reg}(\Sigma)$ as usual, whether there exists $\sigma \in R$ such that $u(\tau) \sqsubseteq v(\tau)$ and $u(\tau') \sqsubseteq v(\tau')$ for all decompositions $\sigma = \tau.\tau'$. In other words, σ is both a direct and a codirect solution.

[1] PEP is undecidable if we allow constraint sets R outside $\mathsf{Reg}(\Sigma)$ [2]. Other extensions, like $\exists x \in R_1 : \forall y \in R_2 : u(xy) \sqsubseteq v(xy)$, for $R_1, R_2 \in \mathsf{Reg}(\Sigma)$, have been shown undecidable [6].

Note that $\mathsf{PEP}_{\text{co\&dir}}$ has no R' parameter (or, equivalently, has $R' = \Sigma^*$) and requires directness and codirectness at all positions. However, this restricted combination is already undecidable:

Theorem 6.3 (Undecidability). $\mathsf{PEP}_{\text{co\&dir}}$ *is* Σ^0_1*-complete.*

Membership in Σ^0_1 is clear and we prove hardness by reducing from the Reachability Problem for length-preserving semi-Thue systems. The undecidability is linked to relying on *different* embeddings of $u(\sigma)$ in $v(\sigma)$ for the directness and codirectness. In contrast, for $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ we need to consider only the leftmost embedding of $u(\sigma)$ in $v(\sigma)$.

A semi-Thue system $S = (\Upsilon, \Delta)$ has a finite set $\Delta \subseteq \Upsilon^* \times \Upsilon^*$ of string rewrite rules over some alphabet Υ, written $\Delta = \{l_1 \to r_1, \ldots, l_k \to r_k\}$. The one-step rewrite relation $\to_\Delta \subseteq \Upsilon^* \times \Upsilon^*$ is defined as usual with $x \to_\Delta y \stackrel{\text{def}}{\Leftrightarrow} x = zlz'$ and $y = zrz'$ for some rule $l \to r$ in Δ and strings z, z' in Υ^*. We write $x \xrightarrow{m}_\Delta y$ and $x \xrightarrow{*}_\Delta y$ when x can be rewritten into y by a sequence of m (respectively, any number, possibly zero) rewrite steps.

The *Reachability Problem* for semi-Thue systems is "Given $S = (\Upsilon, \Delta)$ and two regular languages $P_1, P_2 \in \mathsf{Reg}(\Upsilon)$, is there $x \in P_1$ and $y \in P_2$ s.t. $x \xrightarrow{*}_\Delta y$?". It is well-known (or easy to see by encoding Turing machines in semi-Thue systems) that this problem is undecidable (in fact, Σ^0_1-complete) even when restricted to *length-preserving systems*, i.e., systems where $|l| = |r|$ for all rules $l \to r \in \Delta$.

We now construct a many-one reduction to $\mathsf{PEP}_{\text{co\&dir}}$. Let $S = (\Upsilon, \Delta)$, P_1, P_2 be a length-preserving instance of the Reachability Problem. W.l.o.g., we assume $\varepsilon \notin P_1$ and we restrict to reachability via an even and non-zero number of rewrite steps. With any such instance we associate a $\mathsf{PEP}_{\text{co\&dir}}$ instance $u, v : \Sigma^* \to \Gamma^*$ with $R \in \mathsf{Reg}(\Sigma)$ such that the following correctness property holds:

$$\begin{aligned} &\exists x \in P_1,\ \exists y \in P_2,\ \exists m \text{ s.t. } x \xrightarrow{m}_\Delta y \text{ (and } m > 0 \text{ is even)} \\ \text{iff } &\exists \sigma \in R \text{ s.t. } u(\tau) \sqsubseteq v(\tau) \text{ and } u(\tau') \sqsubseteq v(\tau') \text{ for all decompositions } \sigma = \tau\tau'. \end{aligned} \tag{CP}$$

The reduction uses letters like $\mathtt{a}$, $\mathtt{b}$ and $\mathtt{c}$ taken from Υ, and adds † as an extra letter. We use six copies of each such "plain" letter. These copies are obtained by priming and double-priming letters, and by overlining. Hence the six copies of $\mathtt{a}$ are $\mathtt{a}, \mathtt{a}', \mathtt{a}'', \overline{\mathtt{a}}, \overline{\mathtt{a}'}, \overline{\mathtt{a}''}$. As expected, for a "plain" word (or alphabet) x, we write x' and $\overline{x}$ to denote a version of x obtained by priming (respectively, overlining) all its letters. Formally, letting $\Upsilon_\dagger$ being short for $\Upsilon \cup \{\dagger\}$, one has $\Sigma = \Gamma \stackrel{\text{def}}{=} \Upsilon_\dagger \cup \Upsilon'_\dagger \cup \Upsilon''_\dagger \cup \overline{\Upsilon_\dagger} \cup \overline{\Upsilon'_\dagger} \cup \overline{\Upsilon''_\dagger}$.

We define and explain the reduction by running it on the following example:

$$\Upsilon = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\} \text{ and } \Delta = \{\mathtt{ab} \to \mathtt{bc},\ \mathtt{cc} \to \mathtt{aa}\}. \tag{S_{exmp}}$$

Assume that $\mathtt{abc} \in P_1$ and $\mathtt{baa} \in P_2$. Then $P_1 \xrightarrow{*}_\Delta P_2$ since $\mathtt{abc} \xrightarrow{*}_\Delta \mathtt{baa}$ as witnessed by the following (even-length) derivation π = "$\mathtt{abc} \to_\Delta \mathtt{bcc} \to_\Delta \mathtt{baa}$". In our reduction, a rewrite step like "$\mathtt{abc} \to_\Delta \mathtt{bcc}$" appears in the PEP solution σ as the letter-by-letter interleaving $\mathtt{a}\overline{\mathtt{b}}\mathtt{b}\overline{\mathtt{c}}\mathtt{c}\overline{\mathtt{c}}$, denoted $\begin{smallmatrix}\mathtt{abc}\\ \mathtt{\ bcc}\end{smallmatrix}$, of a plain string and an overlined copy of a same-length string.

Write $T_{\blacktriangleright}(\Delta)$, or just $T_{\blacktriangleright}$ for short, for the set of all $x \quad y$ such that $x \to_\Delta y$. Obviously, and since we are dealing with length-preserving systems, $T_{\blacktriangleright}$ is a regular language, as seen by writing it as $T_{\blacktriangleright} = \left(\sum_{a\in\Upsilon} a\overline{a}\right)^*.\{l \quad r \mid l \to r \in \Delta\}.\left(\sum_{a\in\Upsilon} a\overline{a}\right)^*$, where $\{l \quad r \mid l \to r \in \Delta\}$ is a finite, hence regular, language.

$T_{\blacktriangleright}$ accounts for odd-numbered steps. For even-numbered steps like $\mathtt{bcc} \to_\Delta \mathtt{baa}$ in π above, we use the symmetric $\overline{\mathtt{b}}\overline{\mathtt{b}}\mathtt{a}\overline{\mathtt{a}}\mathtt{c}\overline{\mathtt{c}}$, i.e., $\mathtt{baa} \quad \mathtt{bcc}$. Here too $T_{\blacktriangleleft} \stackrel{\text{def}}{=} \{y \quad x \mid x \to_\Delta y\}$ is regular. Finally, a derivation π of the general form $x_0 \to_\Delta x_1 \to_\Delta x_2 \ldots \to_\Delta x_{2k}$, where $K \stackrel{\text{def}}{=} |x_0| = \ldots = |x_{2k}|$, is encoded as a solution σ_π of the form $\sigma_\pi = \rho_0\sigma_1\rho_1\sigma_2\ldots\rho_{2k-1}\sigma_{2k}\rho_{2k}$ that alternates between the encodings of steps (the σ_i's) in $T_{\blacktriangleright} \cup T_{\blacktriangleleft}$, and *fillers*, (the ρ_i's) defined as follows:

$$\sigma_i \stackrel{\text{def}}{=} \begin{cases} x_{i-1} \quad x_i \text{ for odd } i, \\ x_i \quad x_{i-1} \text{ for even } i, \end{cases} \qquad \rho_0 \stackrel{\text{def}}{=} x_0'' \quad \dagger''^K, \qquad \rho_{2k} \stackrel{\text{def}}{=} x_{2k}'' \quad \dagger''^K, \qquad \rho_i \stackrel{\text{def}}{=} \begin{cases} \dagger'^K \quad x_i' \text{ for odd } i, \\ x_i' \quad \dagger'^K \text{ for even } i \neq 0, 2k. \end{cases}$$

Note that the extremal fillers ρ_0 and ρ_{2k} use double-primed letters, when the internal fillers use primed letters. Continuing our example, the σ_π associated with the derivation $\mathtt{abc} \to_\Delta \mathtt{bcc} \to_\Delta \mathtt{baa}$ is

$$\sigma_\pi = \underbrace{\mathtt{a}''\overline{\dagger''}\mathtt{b}''\overline{\dagger''}\mathtt{c}''\overline{\dagger''}}_{\mathtt{a}''\mathtt{b}''\mathtt{c}'' \quad \dagger''\dagger''\dagger''} \underbrace{\mathtt{a}\overline{\mathtt{b}}\mathtt{b}\overline{\mathtt{c}}\mathtt{c}\overline{\mathtt{c}}}_{\mathtt{abc} \quad \mathtt{bcc}} \underbrace{\dagger'\overline{\mathtt{b}'}\dagger'\overline{\mathtt{c}'}\dagger'\overline{\mathtt{c}'}}_{\dagger'\dagger'\dagger' \quad \mathtt{b}'\mathtt{c}'\mathtt{c}'} \underbrace{\mathtt{b}\overline{\mathtt{b}}\mathtt{a}\overline{\mathtt{c}}\mathtt{a}\overline{\mathtt{c}}}_{\mathtt{baa} \quad \mathtt{bcc}} \underbrace{\mathtt{b}''\overline{\dagger''}\mathtt{a}''\overline{\dagger''}\mathtt{a}''\overline{\dagger''}}_{\mathtt{b}''\mathtt{a}''\mathtt{a}'' \quad \dagger''\dagger''\dagger''}.$$

The point with primed and double-primed copies is that u and v associate them with different images. Precisely, we define

$$\begin{array}{lllll} u(a) = a, & u(a') = \dagger, & u(\dagger') = \dagger, & u(a'') = \varepsilon, & u(\dagger'') = \varepsilon, \\ v(a) = \dagger, & v(a') = a, & v(\dagger') = w_\Upsilon, & v(a'') = a, & v(\dagger'') = w_\Upsilon, \end{array}$$

where a is any letter in Υ, and where w_Υ is a word listing all letters in Υ. E.g., $w_{\{\mathtt{a},\mathtt{b},\mathtt{c}\}} = \mathtt{abc}$ in our running example. The extremal fillers use special double-primed letters because we want $u(\rho_0) = u(\rho_{2k}) = \varepsilon$ (while v behaves the same on primed and double-primed letters). Finally, overlining is preserved by u and v: $u(\overline{x}) \stackrel{\text{def}}{=} \overline{u(x)}$ and $v(\overline{x}) \stackrel{\text{def}}{=} \overline{v(x)}$.

This ensures that, for $i > 0$, $u(\sigma_i) \sqsubseteq v(\rho_{i-1})$ and $u(\rho_i) \sqsubseteq v(\sigma_i)$, so that a σ_π constructed as above is a direct solution. It also ensures $u(\sigma_i) \sqsubseteq v(\rho_i)$ and $u(\rho_{i-1}) \sqsubseteq v(\sigma_i)$ for all $i > 0$, so that σ_π is also a codirect solution. One can check it on our running example by writing $u(\sigma_\pi)$ and $v(\sigma_\pi)$ alongside:

$$\begin{array}{rl} \sigma_\pi = & \overbrace{\mathtt{a}''\overline{\dagger''}\mathtt{b}''\overline{\dagger''}\mathtt{c}''\overline{\dagger''}}^{\rho_0} \; \overbrace{\mathtt{a}\overline{\mathtt{b}}\mathtt{b}\overline{\mathtt{c}}\mathtt{c}\overline{\mathtt{c}}}^{\sigma_1} \; \overbrace{\dagger'\overline{\mathtt{b}'}\dagger'\overline{\mathtt{c}'}\dagger'\overline{\mathtt{c}'}}^{\rho_1} \; \overbrace{\mathtt{b}\overline{\mathtt{b}}\mathtt{a}\overline{\mathtt{c}}\mathtt{a}\overline{\mathtt{c}}}^{\sigma_2} \; \overbrace{\mathtt{b}''\overline{\dagger''}\mathtt{a}''\overline{\dagger''}\mathtt{a}''\overline{\dagger''}}^{\rho_2} \\ \hline u(\sigma_\pi) = & \mathtt{a}\overline{\mathtt{b}}\mathtt{b}\overline{\mathtt{c}}\mathtt{c}\overline{\mathtt{c}} \quad \dagger\dagger\overline{\dagger}\dagger\dagger\overline{\dagger}\dagger\dagger \quad \mathtt{b}\overline{\mathtt{b}}\mathtt{a}\overline{\mathtt{c}}\mathtt{a}\overline{\mathtt{c}} \\ v(\sigma_\pi) = & \mathtt{a}\,\overline{\mathtt{abc}}\,\mathtt{b}\,\overline{\mathtt{abc}}\,\mathtt{c}\,\overline{\mathtt{abc}} \quad \dagger\dagger\overline{\dagger}\dagger\dagger\overline{\dagger}\dagger\dagger \quad \mathtt{abc}\,\overline{\mathtt{b}}\,\mathtt{abc}\,\overline{\mathtt{c}}\,\mathtt{abc}\,\overline{\mathtt{c}} \quad \dagger\dagger\overline{\dagger}\dagger\dagger\overline{\dagger}\dagger\dagger \quad \mathtt{b}\,\overline{\mathtt{abc}}\,\mathtt{a}\,\overline{\mathtt{abc}}\,\mathtt{a}\,\overline{\mathtt{abc}} \end{array}$$

There remains to define R. Since $\rho_0 \in \left(\Upsilon''\overline{\dagger''}\right)^+$, since $\sigma_i \in T_{\blacktriangleright}$ for odd i, etc., we let

$$R \stackrel{\text{def}}{=} \left(\Upsilon''\overline{\dagger''}\right)^+.T_{\blacktriangleright}^{\cap P_1}.\left(\dagger'\overline{\Upsilon'}\right)^+.\left(T_{\blacktriangleleft}.\left(\Upsilon'\overline{\dagger'}\right)^+.T_{\blacktriangleright}.\left(\dagger'\overline{\Upsilon'}\right)^+\right)^*.T_{\blacktriangleleft}^{\cap P_2}.\left(\Upsilon''\overline{\dagger''}\right)^+, \qquad (1)$$

where $T_{\blacktriangleright}^{\cap P_1} \stackrel{\text{def}}{=} \{x \quad y \mid x \rightarrow_\Delta y \wedge x \in P_1\} = T_{\blacktriangleright} \cap \{x \quad y \mid x \in P_1 \wedge |x| = |y|\}$ is clearly regular when P_1 is, and similarly for $T_{\blacktriangleleft}^{\cap P_2} \stackrel{\text{def}}{=} \{y \quad x \mid x \rightarrow_\Delta y \wedge y \in P_2\}$. Since $\sigma_\pi \in R$ when π is an even-length derivation from P_1 to P_2, we deduce that the left-to-right implication in (CP) holds.

We refer to the full version of this paper at `arxiv.org/abs/1109.1691` for a proof that the right-to-left implication also holds, which concludes the proof of Theorem 6.3.

7 Concluding Remarks

We introduced partial directness in Post Embedding Problems and proved the decidability of $\mathsf{PEP}_{\text{dir}}^{\text{partial}}$ by showing that an instance has a solution if, and only if, it has a solution of length bounded by a computable function of the input. This generalizes and simplifies earlier proofs for PEP and $\mathsf{PEP}_{\text{dir}}$. The added generality is non-trivial and leads to decidability for UCST, or UCS (that is, unidirectional channel systems) extended with tests [10]. The simplification lets us deal smoothly with counting or universal versions of the problem. Finally, we showed that *combining* directness and codirectness constraints leads to undecidability.

References

1. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Information and Computation 127(2), 91–101 (1996)
2. Chambart, P., Schnoebelen, P.: Post Embedding Problem Is Not Primitive Recursive, with Applications to Channel Systems. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 265–276. Springer, Heidelberg (2007)
3. Chambart, P., Schnoebelen, P.: Mixing Lossy and Perfect Fifo Channels. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 340–355. Springer, Heidelberg (2008)
4. Chambart, P., Schnoebelen, P.: The ω-Regular Post Embedding Problem. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 97–111. Springer, Heidelberg (2008)
5. Chambart, P., Schnoebelen, P.: The ordinal recursive complexity of lossy channel systems. In: Proc. LICS 2008, pp. 205–216. IEEE Comp. Soc. Press (2008)
6. Chambart, P., Schnoebelen, P.: Computing Blocker Sets for the Regular Post Embedding Problem. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 136–147. Springer, Heidelberg (2010)
7. Chambart, P., Schnoebelen, P.: Pumping and Counting on the Regular Post Embedding Problem. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part II. LNCS, vol. 6199, pp. 64–75. Springer, Heidelberg (2010)
8. Fairtlough, M., Wainer, S.S.: Hierarchies of provably recursive functions. In: Buss, S. (ed.) Handbook of Proof Theory. Studies in Logic, ch. 3, vol. 137, pp. 149–207. Elsevier Science (1998)
9. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science 256(1-2), 63–92 (2001)
10. Jančar, P., Karandikar, P., Schnoebelen, P.: Unidirectional channel systems can be tested (in preparation, 2012)
11. Muscholl, A.: Analysis of Communicating Automata. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 50–57. Springer, Heidelberg (2010)
12. Schmitz, S., Schnoebelen, P.: Multiply-Recursive Upper Bounds with Higman's Lemma. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 441–452. Springer, Heidelberg (2011)

On the Advice Complexity of the Set Cover Problem*

Dennis Komm[1], Richard Královič[1], and Tobias Mömke[2]

[1] Department of Computer Science, ETH Zurich, Switzerland
{dennis.komm,richard.kralovic}@inf.ethz.ch
[2] School of Computer Science and Communication,
KTH Royal Institute of Technology, Sweden
moemke@kth.se

Abstract. Recently, a new approach to get a deeper understanding of online computation has been introduced: the study of the *advice complexity* of online problems. The idea is to measure the information that online algorithms need to be supplied with to compute high-quality solutions and to overcome the drawback of not knowing future input requests. In this paper, we study the advice complexity of an online version of the well-known set cover problem introduced by Alon et al.: for a ground set of size n and a set family of m subsets of the ground set, we obtain bounds in both n and m. We prove that a linear number of advice bits is both sufficient and necessary to perform optimally. Furthermore, for any constant c, we prove that $n - c$ bits are enough to construct a c-competitive online algorithm and this bound is tight up to a constant factor (only depending on c). Moreover, we show that a linear number of advice bits is both necessary and sufficient to be optimal with respect to m, as well. We further show lower and upper bounds for achieving c-competitiveness also in m.

1 Introduction and Preliminaries

The idea of online algorithms is both a natural concept in practical applications and of great theoretical interest. In this model, the algorithm designer faces the problem that an algorithm A has to read the input instance of some problem piecewise and has to create parts of the output before knowing the whole input string. Furthermore, A is not allowed to revoke any pieces of submitted output.

Online scenarios and online algorithms have been studied extensively (for an overview, we refer the reader to the standard literature [1,7,11,13]). Similar to the concept of the approximation ratio of algorithms dealing with (hard) offline problems, *competitive analysis* (introduced in 1985 by Sleator and Tarjan [17]) is usually used to measure the quality of online algorithms (ignoring the runtime of the algorithm): for any instance I of some online minimization problem, we define the competitive ratio of A on I as the fraction of the cost of A on I and

* This work was partially supported by ERC Advanced investigator grant 226203.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 241–252, 2012.

the cost of an optimal solution $\textsc{Opt}(I)$. Here, $\textsc{Opt}$ is an *offline* algorithm that sees the whole input in advance. Clearly, we may see the competitive ratio as a value that tells us what we forfeit for not knowing the future. When dealing with the *advice complexity* of an online problem, we are interested in getting a deeper understanding of *what* it is we pay for. A straightforward answer to this question is "the remainder of the instance", but we want to measure the essence of what makes the problem hard with respect to online computation.

In the model used throughout this paper, we consider online algorithms that are able to receive extra information from some *advice tape* that is being created by an oracle $\mathtt{O}$ that knows the whole input. In every time step, an online algorithm $\mathtt{A}$ may sequentially read some piece of information from this tape together with the input. More formally, we are dealing with the following model.

Definition 1 (Online Algorithm with Advice). *Consider an input sequence* $I = (x_1, \ldots, x_n)$. *An* online algorithm $\mathtt{A}$ with advice *computes the output sequence* $\mathcal{A}^\phi = \mathtt{A}^\phi(I) = (y_1, \ldots, y_n)$ *such that* y_i *is computed from* $\phi, x_1, \ldots, x_i$, *where* ϕ *is the content of the advice tape, i. e., an infinite binary sequence.* $\mathtt{A}$ *is* c-competitive with advice complexity $s(n)$ *if there exists a constant* α *such that, for every* n *and for each input sequence* I *of length at most* n, *there exists some* ϕ *such that* $\mathrm{cost}(\mathtt{A}^\phi(I)) \leq c \cdot \mathrm{cost}(\textsc{Opt}(I)) + \alpha$ *and at most the first* $s(n)$ *bits of* ϕ *have been accessed during the computation of* $\mathtt{A}^\phi(I)$. *Moreover, if* $\alpha = 0$, $\mathtt{A}$ *is called* strictly c*-competitive. An algorithm is* optimal *if it is strictly* 1*-competitive.*

In this paper, we consider *strictly* competitive algorithms only, when talking about both lower and upper bounds.

The advice complexity was introduced by Dobrev, Pardubská, and Královič [9] in 2008 and since then a revised version was used to study various online problems. For a detailed survey on this topic we refer to Hromkovič, Královič, and Královič [12]. Previous to this paper, the advice complexity of various problems was analyzed: the paging problem and the disjoint path allocation problem [5], the job shop scheduling problem [5,14], metrical task systems [10], the k-server problem [3,10], and the knapsack problem [4]. There are interesting connections between computing with advice and randomization [3,14].

The study of advice complexity has an interesting connection to recent developments in the field of dynamic data structures, where the use of advice bits was proposed as a lower bound technique [16,8].

A Note on the Model

As usual, in order to construct hard instances for the online algorithms studied, we consider an *adversary* $\textsc{Adv}$ that constructs the input in a malicious way [6,7,11,13]. As mentioned, we add another player to the game that is classically played between an online algorithm $\mathtt{A}$ and $\textsc{Adv}$: the oracle $\mathtt{O}$ that collaborates with $\mathtt{A}$ by giving help using the advice tape. Note that, similar to randomized online algorithms, we can think of an online algorithm $\mathtt{A}$ with advice as an algorithm that chooses, depending on the advice it is given, from a set

of deterministic strategies denoted by Alg(A). Another view we might impose on *computations with advice* is that an optimal random choice is supplied by O for every input.

Observation 1 [14]. *Let* A *be an online algorithm that uses b bits of advice. Then* Adv *can handle* A *as 2^b different deterministic online algorithms without advice. In particular,* Adv *knows each of the 2^b algorithms.*

This observation enables us to use two different techniques to show lower bounds on the number of advice bits required to obtain a certain output quality; for instance, Böckenhauer at al. [5] proved the hardness by a combinatorial argument: suppose that there is an online algorithm A that uses b bits of advice. Then there must be an advice tape such that reading the first b bits enables A to distinguish a class of relevant inputs sufficiently for the required properties of A. The goal is then to show that there are different inputs that cannot be distinguished sufficiently. Using Observation 1, we are able to design an adversary Adv that creates an input instance such that all deterministic algorithms from Alg(A) fail to meet all required properties. We assume that each of the 2^b choices leads to a valid algorithm for the considered input. Otherwise the number of valid choices decreases, which can only strengthen the hardness results.

In the following, we introduce some observations that are crucial for techniques to prove lower bounds on the number of advice bits necessary to achieve some specific competitive ratio. Ben-David et al. [6] introduced three different types of adversaries that are all equally powerful when considering deterministic online algorithms, but are variably powerful for randomized online algorithms: (i) the *oblivious* adversary, (ii) the *adaptive online* adversary, and (iii) the *adaptive offline* adversary. Originally, a model was introduced that implies that Adv knows O [5,3]; thus, we are dealing with a highly powerful adversary. When proving lower bounds, it was usually shown that, for some advice string of some fixed length, there exist different inputs within one class of inputs that cannot be distinguished sufficiently to achieve some specific competitive ratio. However, Adv must know the (at least) two inputs that correspond to one advice string causing A to fail being given one of them, i. e., Adv has to know how O prepares the advice strings to select the concrete input. Formally, it was shown that

$$\forall\ (\mathtt{A}, \mathtt{O})\ \exists\ \textsc{Adv}\ \text{creating}\ I\ \text{such that}\ \mathrm{cost}(\mathtt{A}^{\phi}(I)) > c \cdot \mathrm{cost}(\textsc{Opt}(I)). \qquad (1)$$

So here, the oracle is fixed (known to Adv) and we show that for every algorithm and oracle such an adversary exists.

Interestingly, some of the statements we make throughout this paper are formulated in a way such that Adv does not have to know which advice string is given; it is sufficient if it knows how A behaves on all possible advice strings (i. e., this is indeed an oblivious adversary). Thus, we show that

$$\forall\ \mathtt{A}\ \exists\ \textsc{Adv}\ \text{creating}\ I\ \text{such that}\ \forall\ \mathtt{O}\ \text{we have}\ \mathrm{cost}(\mathtt{A}^{\phi}(I)) > c \cdot \mathrm{cost}(\textsc{Opt}(I)). \quad (2)$$

While it seems obvious that (2) is stronger than (1), both models are indeed equally powerful. Suppose that Adv merely knows A. It then may *learn* which one

is the *strongest* oracle for *all* inputs [15]. There must exist one such oracle that is never worse than the other possible ones on every input (towards contradiction, suppose that there exist I_1 and I_2 such that $\mathtt{O}_1$ gives a strictly stronger advice for I_1 while $\mathtt{O}_2$ gives a better advice for I_2, for a fixed algorithm A. Then, we can construct an oracle O that simulates $\mathtt{O}_1$ on I_1 and $\mathtt{O}_2$ on I_2). This way, Adv can determine the worst input for the best oracle. We therefore can safely say that Adv, as we used it in this paper, is really *oblivious* (with the restriction that it knows the number b of advice bits A uses).

Throughout this paper, for any set Y, by $\mathcal{P}(Y)$ we denote the power set of Y; by $\log x$ we denote the logarithm of x with base 2. Note that, due to space constraints, we have to omit some proofs, which can be found in the technical report [15].

The Set Cover Problem

In the following, we consider an online version of the set cover problem, SetCover for short, introduced by Alon et al. [2].

Definition 2 (SetCover). *Given a* ground set X *of size* n*, a set of* requests $X' \subseteq X$*, and a family* $\mathcal{S} \subseteq \mathcal{P}(X)$ *of size* m*, a feasible solution for* SetCover *is any subset* $\{S_1, \ldots, S_k\}$ *of* $\mathcal{S}$ *such that*

$$\bigcup_{i=1}^{k} S_i \supseteq X'.$$

The aim is to minimize k*, i. e., to use as few sets as possible. In the online version of this problem, the elements of* X' *arrive successively one by one in consecutive time steps. An online algorithm* A *solves* SetCover *if, immediately after each yet uncovered request* j*, it specifies a set* S_i *such that* $j \in S_i$*.*

Note that Alon et al. constructed a deterministic $\mathcal{O}(\log m \log n)$-competitive algorithm that even works for the weighted version of SetCover[1] [2]. In the following, we give bounds on the advice complexity in both the size of the ground set X and the size of the family $\mathcal{S}$. Obviously, $n = |X|$ is an upper bound on the number of requests. Furthermore, note that we may assume that $\mathcal{S}$ does not contain any sets that are subsets of any other set from $\mathcal{S}$ (if not, sets that are contained in other sets may be removed from the input by a preprocessing step; since the sets are unweighted, it is never worse to prefer supersets). From this, it directly follows, due to Sperner's theorem [18], that we have

$$|\mathcal{S}| \leq \binom{n}{\lfloor \frac{n}{2} \rfloor} =: N(n),$$

which trivially implies that m may be exponential in n. However, due to the hardness of the vertex cover problem (which is a sub-problem of SetCover) we

[1] Here, every member of $\mathcal{S}$ has a non-negative weight, and the objective is to minimize the sum of the weights of all items chosen.

know that also instances where $m \in \mathcal{O}(n)$ may be hard in the offline case. This motivates the study of advice depending on both n and m.

Sometimes, when we need to communicate a number x in a self-delimiting way, we use a well-known technique that enables us to use at most $2\lceil \log \lceil \log x \rceil \rceil + \lceil \log x \rceil$ bits (see, e. g., [5]).

Organization of this Paper

In Section 2, we measure the advice complexity as a function of the size of the set of elements to cover, that is, $n = |X|$. We show that linear advice is both sufficient and necessary to create optimal output. As for the trade-off between an achievable constant competitive ratio c and the number of advice bits, we show an upper bound of $n - c + 2$ and a lower bound that matches up to some factor depending on c. In Section 3, we give bounds in the size m of $\mathcal{S}$. Here, we also show that linear advice is necessary and sufficient to be optimal; this is tight up to a factor of $(\log 3)/3$. We prove a lower bound of $\log m/c - 1$ for achieving c-competitiveness and an upper bound of roughly $m - c \cdot m/\log m$.

2 Bounds Measured in the Size of X

First, let us analyze the *information complexity* of the problem, that is, the advice complexity for calculating an optimal solution [12]. In the following, we show that a number of advice bits linear in $|X|$ is sufficient to create an optimal output by a very easy argument.

Theorem 1. *There exists an optimal online algorithm* `A` *with advice that uses* $n - 1$ *advice bits.*

We omit the proof due to a lack of space; see [15]. Although the above algorithm is trivial, it is interesting to note that we are able to complement this result with an almost matching lower bound to show that a linear number of advice bits is also necessary to be optimal.

Theorem 2. *At least* $\log(N(n-1)) > n - \frac{1}{2}\log(n-1) - 3$ *bits are necessary for any online algorithm* `A` *with advice to achieve optimality, for any even* n.

Proof. Let n be even. Consider the set $\mathcal{S}$ that contains all subsets of X of size exactly $n/2$. Clearly, there are exactly $N(n)$ such sets. Now `Adv` requests $n/2$ items, starting with one fixed item x_1 (after which `A` has to start with choosing a set from $\mathcal{S}$). Clearly, one single set from $\mathcal{S}$ is sufficient to cover all requests. Since `A` knows that x_1 is included in the set it has to choose, there are

$$\binom{n-1}{n/2-1}$$

remaining candidate sets. Observe that, since n is even, it holds that $n/2 - 1 = \lfloor (n-1)/2 \rfloor$. Consequently, there are exactly $N(n-1)$ sets left from which `A` has to select one.

Therefore, A needs to distinguish between $N(n-1)$ different advices to distinguish this many sets and if it reads strictly less than $\log(N(n-1))$ bits, this merely enables A to choose among strictly less than $N(n-1)$ strategies which, by Observation 1, is equivalent to choosing among this many deterministic algorithms.

By the pigeonhole principle, there exist two distinct sets S_1 and S_2 (i. e., inputs which both contain x_1, but differ in at least one element) for which A chooses the same deterministic strategy $A \in \mathrm{Alg}(\mathtt{A})$. Clearly, A cannot be optimal in both cases. Using Stirling's approximation, we get

$$\log(N(n-1)) \geq \log\left(\frac{4^{(n-1)/2}}{2\sqrt{(n-1)\pi/2}}\right) > n - \frac{1}{2}\log(n-1) - 3,$$

which finishes the proof. □

The next question we are dealing with is how many bits are necessary and sufficient to achieve c-competitiveness.

Theorem 3. *There exists a c-competitive online algorithm* A *with advice that uses at most $n - ((c-1)k+1) + \lceil \log k \rceil + 2\lceil \log \lceil \log k \rceil \rceil$ bits of advice where k the size of an optimal solution.*

Proof. Suppose that an optimal solution $\mathcal{O}pt$ covers all $|X'|$ requests with k elements from $\mathcal{S}$. The number k can be communicated to A in a self-delimiting way using $\lceil \log k \rceil + 2\lceil \log \lceil \log k \rceil \rceil$ advice bits. Note that A may use at most $c \cdot \mathrm{cost}(\mathcal{O}pt) = \mathrm{cost}(\mathcal{O}pt) + (c-1)k$ sets to be c-competitive.

As in the proof of Theorem 1, O writes the characteristic function of X' onto the advice tape. Recall that A knows the first element from X' that is requested before it has to make any decision. Since A may use $(c-1)k$ additional sets in comparison to $\mathcal{O}pt$, it only needs to read the first $n-(c-1)k-1$ bits of advice, compute the optimal solution $\mathcal{O}pt^*$ for this sub-instance (which, of course, has a smaller cost than $\mathcal{O}pt$) and take one additional set for every requested element not covered by $\mathcal{O}pt^*$. □

Note that, for any $k \geq 5$, $c \leq ck + 1 - k - \lceil \log k \rceil - 2\lceil \log \lceil \log k \rceil \rceil$ follows from

$$c \geq \frac{k + \log k + 2\log\log k + 2}{k-1},$$

which is true for any $c \geq 3$. Together with Theorem 3, this implies the following corollary.

Corollary 1. *For any $c \geq 3$, there exists a c-competitive online algorithm* A *with advice for* SetCover *that uses at most $n - c$ bits of advice and an optimal solution that uses more than five sets.* □

Next, generalizing the idea from the proof of Theorem 2, we show a lower bound on the number of advice bits required for any online algorithm that achieves c-competitiveness. To do so, we again consider an adversary Adv that plays against some online algorithm A with advice, i. e., against 2^b deterministic algorithms at once (see Observation 1).

Theorem 4. *Any deterministic online algorithm with advice that reads at most*

$$(n - c^2 - 2c - 1)\frac{\log\left(\frac{c+1}{c}\right)}{c^2 + c},$$

bits of advice is guaranteed to be strictly worse than c-competitive.

Proof. Let n be a multiple of $c+1$. In the following, for any given c, $\mathcal{S}$ consists of all possible sets of size $n/(c+1)$ in $\mathcal{P}(X)$. Similar to the previous discussion, an adversary ADV chooses exactly $n/(c+1)$ items such that there exists a unique optimal solution that consists of exactly one set. Thus, a c-competitive algorithm is allowed to choose c sets at most.

Let A be an online algorithm that uses b bits of advice. We will determine a number b such that all algorithms in Alg(A) fail to be c-competitive, that is, are forced to choose at least $c+1$ sets. To this end, ADV proceeds in rounds, where in each round all algorithms from Alg(A) are forced to choose an additional set whereas there is an optimal solution that consists of only one single set for the whole instance.

There are $c+1$ rounds numbered $0, \ldots, c$. At the end of round i, ADV has to ensure that all algorithms have chosen at least $i+1$ sets so far. In each time step, the algorithms in Alg(A) are partitioned into those that already chose an additional set in the current round (these algorithms are said to be *out for this round*) and those that did not yet choose an additional set.

ADV creates an input instance $I = (i_1, \ldots, i_{n/(c+1)})$ that corresponds to exactly one set in $\mathcal{S}$. As in the proof of Theorem 2, the item i_1 is fixed as x_1 (in round 0, i. e., before A produces any output) such that each algorithm in Alg(A) has to choose a set containing x_1. Subsequently, ADV enters round 1 and it chooses an item such that as many algorithms from Alg(A) as possible are out for this round, because they do not have the corresponding item covered yet. ADV continues in this fashion until, eventually, round 1 is finished since all algorithms are out for this round. The remaining rounds work analogously.

Let p_0 be 1 and $p_1, \ldots, p_c$ be natural numbers such that ADV chooses p_i elements to finish round i, i. e., ADV chooses $p_i - p_{i-1}$ many items to force all algorithms in Alg(A) to be out for round i. At the beginning of round r, there are, in total, 2^b subsets of $\mathcal{S}$ that were determined by the algorithms from Alg(A) each of which contains the items of the earlier rounds. We can assume that there is no uncovered item; otherwise, ADV requests exactly this one thus finishing round r. Clearly, we do not have to consider any algorithms that already chose $r+1$ sets in previous rounds. Therefore, ADV only deals with algorithms that have chosen a total number of exactly r sets and, thus, at most $rn/(c+1)$ items. From these items, p_{r-1} are fixed already, because they were requested in previous rounds. Thus, there are $rn/(c+1) - p_{r-1}$ items that may be requested to make one particular algorithm out for round r. Summing over all 2^b algorithms, the number of occurrences of all not yet requested items is at most

$$2^b \left(\frac{rn}{c+1} - p_{r-1} \right).$$

On the other hand, in the first time step of round r, Adv can choose from $n-p_{r-1}$ items from X. It follows that the average number of occurrences of one selectable item is at most

$$\frac{2^b\left(\frac{rn}{c+1}-p_{r-1}\right)}{n-p_{r-1}}.$$

Therefore, the least frequently covered item has already been covered by at most that number of different algorithms in Alg(A). This is exactly the item $i_{p_{r-1}+1}$ that Adv chooses in this time step. Since any algorithm is out for this round if it has chosen a family of sets not containing this item, in the subsequent steps, we only have to focus on the remaining algorithms in Alg(A).

More generally, Adv can reduce the number of algorithms that are not out for round r by a fraction of

$$\frac{\frac{rn}{c+1}-j}{n-j} \tag{3}$$

or more, for $j \geq p_{r-1}$, when taking the $(j+1)$-th element of the instance I. Hence, let $s \leq rn/(c+1)+1$ be a natural number such that

$$2^b \prod_{j=p_{r-1}}^{s-1} \frac{\frac{rn}{c+1}-j}{n-j} < 1. \tag{4}$$

Then, it follows that $p_r \leq s$. Note that, since $r < c+1$, $n > rn/(c+1) \geq s-1$ holds in (4), thus the fraction is well-defined. Moreover, there exists s such that (4) holds, e. g., $s = rn/(c+1)+1$. Therefore, the value of (4) is at most

$$2^b \prod_{j=p_{r-1}}^{s-1} \frac{\frac{rn}{c+1}}{n} = 2^b \left(\frac{r}{c+1}\right)^{s-p_{r-1}}. \tag{5}$$

Hence, to satisfy (4), it is sufficient to choose s such that

$$2^b \left(\frac{r}{c+1}\right)^{s-p_{r-1}} < 1 \iff s - p_{r-1} > b\,\frac{1}{\log(c+1)-\log r}$$

Following this, we can choose

$$s := \left\lfloor \frac{b}{\log(c+1)-\log r} \right\rfloor + 1 + p_{r-1}. \tag{6}$$

Clearly, Adv succeeds if it uses no more than $n/(c+1)$ items in total, which means that

$$1 + \sum_{r=1}^{c} (p_r - p_{r-1}) \leq \frac{n}{c+1}. \tag{7}$$

Since $p_r \leq s$, (7) is guaranteed by

$$1 + \sum_{r=1}^{c} \left(\left\lfloor \frac{b}{\log\left(\frac{c+1}{r}\right)} \right\rfloor + 1 \right) \leq \frac{n}{c+1}$$

which is implied by

$$b \cdot \sum_{r=1}^{c} \frac{1}{\log\left(\frac{c+1}{r}\right)} \leq \frac{n}{c+1} - c - 1 \iff b \leq \frac{\frac{n}{c+1} - c - 1}{\sum_{r=1}^{c} \frac{1}{\log\left(\frac{c+1}{r}\right)}}$$

which again holds if

$$b \leq \frac{\frac{n-c^2-2c-1}{c+1}}{\frac{c}{\log\left(\frac{c+1}{c}\right)}} = (n - c^2 - 2c - 1)\frac{\log\left(\frac{c+1}{c}\right)}{c^2 + c}.$$

Hence, if b is smaller than claimed in the statement of the theorem, ADV can make sure that any algorithm is worse than c-competitive, finishing the proof. □

Note that, for the instances we constructed in the proof of Theorem 4, there exists a very easy deterministic online algorithm that achieves $(c+1)$-competitiveness. Indeed, it is sufficient to use $c+1$ disjoint sets from $\mathcal{S}$ to cover all items from X. This means that, for such instances, we observe a very interesting threshold: For being a little better, we have to pay with using a linear number of advice bits instead of using no advice at all. Since

$$(n - c^2 - 2c - 1)\frac{\log\left(\frac{c+1}{c}\right)}{c^2 + c} \in \Omega(n),$$

for any constant c, we immediately obtain the following statement.

Corollary 2. *Any online algorithm with advice that achieves a constant competitive ratio c, is required to use a number of advice bits linear in n. Therefore, the upper bound of Theorem 3 is tight up to a constant factor.*

3 Bounds Measured in the Size of $\mathcal{S}$

In the previous section, we measured the advice complexity in the number of elements, $|X|$. Now we look at the online SETCOVER problem from a different perspective by measuring the advice in the number of sets given, $|\mathcal{S}|$. As we discuss at the end of this section, we obtain bounds that are not comparable with those presented in the previous section.

At first, let us consider the advice needed to create an optimal output. Encoding the characteristic function of the sets that are used by an optimal solution immediately gives the following result.

Theorem 5. *There exists an optimal online algorithm* A *that uses m bits of advice.*

As in the previous section, this very naive approach is asymptotically the best we can hope for.

Theorem 6. *Any online algorithm* A *with advice needs at least* $(m \log 3)/3 - 2$ *advice bits to be optimal.*

Proof. For any m, let m' be the largest number that is smaller than or equal to m and that is a multiple of 3, i. e., $m' \geq m - 2$. Moreover, let $n := 4m'/3$. Adv chooses X such that there are $n/4$ items $y_1, \ldots, y_{n/4}$ and $3n/4$ items $x_{i,j}$, $i \in \{1, 2, 3\}$, $j \in \{1, \ldots, n/4\}$. After that, Adv sets $\mathcal{S}$ to contain exactly the following sets. For $k \in \{1, \ldots, n/4\}$, there are three sets $\{y_k, x_{1,k}\}$, $\{y_k, x_{2,k}\}$, and $\{y_k, x_{3,k}\}$. Since $\mathcal{S}$ has to be of size m, Adv adds $m - m'$ many *dummy* sets to $\mathcal{S}$ that each contain one unique item y_j only; these sets are never considered by any optimal solution (which A may assume, because they are subsets of other sets).

Next, Adv requests all items y_i; hence, since each request has to be satisfied, A has to fix $n/4$ sets to cover all items. After that, for every i, Adv requests one of the three items $x_{1,i}$, $x_{2,i}$, and $x_{3,i}$. Note that any combination leads to a distinct optimal solution that consists of exactly $m'/3$ sets. This way, A has to correctly choose one of $3^{m'/3}$ possible families as answers for the first $m'/3$ requests. If, however, A uses less than $\log 3^{m'/3}$ advice bits, then, by applying the pigeonhole principle, there must be two identical advices for two different sequences of correct answers. Thus, Adv can choose the one not chosen by the algorithm. Consequently, a lower bound on the advice complexity is

$$\log 3^{m'/3} \geq \frac{m-2}{3} \log 3 > \frac{m \log 3}{3} - 2$$

which is larger than $m/2$ for m tending to infinity. □

Let us now consider algorithms that aim at achieving c-competitiveness.

Theorem 7. *For any* $c \leq \log m + 1$,

$$m - \frac{(c-1)m}{\lceil \log m \rceil + c - 1} + 1$$

bits are sufficient to achieve c*-competitiveness.*

The proof is omitted due to space constraints, see [15]. We now show a lower bound on yielding c-competitiveness measured in m. This way, we obtain a slightly better lower bound than that one that is directly implied by the lower bound measured in n from the previous section.

Theorem 8. *Any online algorithm* A *with advice needs to read at least* $\lceil \log m/c - 1 \rceil$ *bits of advice to be* c*-competitive.*

This proof is also omitted due to space constraints [15]; the lower bound on the trade-off between c and b is shown in Figure 1.

Compare these results to the ones measured in n from the previous section. As for optimality, in Theorem 6, we had $4m/3 \geq n \geq 4(m-2)/3$, and thus $m \leq (3n+8)/4$ yielding a lower bound of less than $(n+3) \log 3/4$, which is worse than

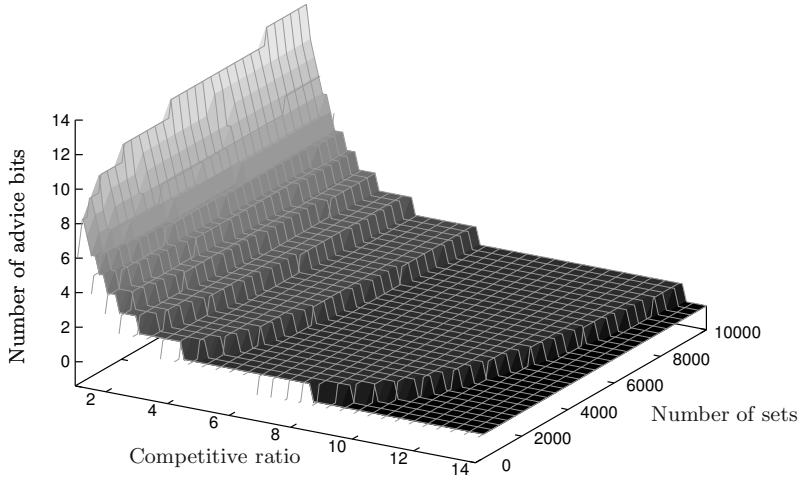


Fig. 1. The competitive ratio $c > 1$ and the advice bits b required depending on m

the one of Theorem 2. On the other hand, in the proof of Theorem 2, we had $m = N(n)$ which means that it merely gives a logarithmic lower bound in m, whereas Theorem 6 gives a linear lower bound. If we look at the trade-off between the advice bits and the competitive ratio, in Theorem 8, we clearly have $m = (\alpha + 1)^c$ and $n = \sum_{i=0}^{c} (\alpha + 1)^i = \frac{(\alpha+1)^{c+1}-1}{\alpha}$. Thus, Theorem 8 yields a lower bound that is logarithmic in n and is therefore worse than the one of Theorem 4. Furthermore, inspecting this theorem, we observe that here $m = \binom{n}{n/(c+1)} \geq (c+1)^{n/(c+1)}$ and the bound of Theorem 8 is better with respect to m and c.

4 Conclusion

In Section 2, we encoded information about the input on the advice tape such that `A` still had to compute the solution itself. Therefore, it was left with solving an $\mathcal{NP}$-hard problem. As common in online computation, this concept of *unconditional hardness* is used to analyze the algorithm's output quality while neglecting its runtime.

On the other hand, the upper bounds with respect to the size of $|\mathcal{S}|$ presented in Section 3 encode information about the *solution*; thus, the resulting algorithms run in polynomial time. In general, it makes sense to analyze the advice complexity of efficient online algorithms. Almost nothing is known about the comparison of online algorithms with advice with unrestricted and restricted runtime, opening an interesting field for further research.

References

1. Albers, S.: Online algorithms: a survey. Mathematical Programming 97(1), 3–26 (2003)

2. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: The online set cover problem. SIAM Journal on Computing 39(2), 361–370 (2009)
3. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the Advice Complexity of the k-Server Problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
4. Böckenhauer, H.-J., Komm, D., Královič, R., Rossmanith, P.: On the Advice Complexity of the Knapsack Problem. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 61–72. Springer, Heidelberg (2012)
5. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the Advice Complexity of Online Problems (Extended Abstract). In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
6. Ben-David, S., Borodin, A., Karp, R.M., Tardos, G., Wigderson, A.: On the power of randomization in on-line algorithms. Algorithmica 11(1), 2–14 (1994)
7. Borodin, A., El-Yaniv, R.: Online computation and Competitive Analysis. Cambridge University Press (1998)
8. Chattopadhyay, A., Edmonds, J., Ellen, F., Pitassi, T.: A little advice can be very helpful. In: Proc. of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012) (to appear, 2012)
9. Dobrev, S., Královič, R., Pardubská, D.: Measuring the problem-relevant information in input. Theoretical Informatics and Applications (RAIRO) 43(3), 585–613 (2009)
10. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theoretical Computer Science 412(24), 2642–2656 (2011)
11. Fiat, A., Woeginger, G.J. (eds.): Online Algorithms 1996. LNCS, vol. 1442. Springer, Heidelberg (1998)
12. Hromkovič, J., Královič, R., Královič, R.: Information Complexity of Online Problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
13. Irani, S., Karlin, A.R.: On online computation. In: Approximation Algorithms for $\mathcal{NP}$-Hard Problems, ch. 13, pp. 521–564 (1997)
14. Komm, D., Královič, R.: Advice complexity and barely random algorithms. Theoretical Informatics and Applications (RAIRO) 45(2), 249–267 (2011)
15. Komm, D., Královič, R., Mömke, T.: On the advice complexity of the set cover problem. Technical Report 738, ETH Zurich (2011)
16. Pătraşcu, M.: Towards polynomial lower bounds for dynamic problems. In: Proc. of the 42nd ACM Symposium on Theory of Computing (STOC 2010), pp. 603–610 (2010)
17. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28(2), 202–208 (1985)
18. Sperner, E.: Ein Satz über Untermengen einer endlichen Menge. Mathematische Zeitschrift 27(1), 544–548 (1928)

Constraint Satisfaction with Counting Quantifiers*

Florent Madelaine[1], Barnaby Martin[2,**], and Juraj Stacho[3,***]

[1] Clermont Université, Université d'Auvergne,
LIMOS, BP 10448, F-63000 Clermont-Ferrand, France
[2] School of Engineering and Computing Sciences, Durham University,
Science Laboratories, South Road, Durham DH1 3LE, UK
[3] DIMAP and Mathematics Institute,
University of Warwick, Coventry CV4 7AL, UK

Abstract. We initiate the study of *constraint satisfaction problems* (CSPs) in the presence of counting quantifiers, which may be seen as variants of CSPs in the mould of *quantified CSPs* (QCSPs).

We show that a **single** counting quantifier strictly between $\exists^{\geq 1} := \exists$ and $\exists^{\geq n} := \forall$ (the domain being of size n) already affords the maximal possible complexity of QCSPs (which have **both** $\exists$ and $\forall$), being Pspace-complete for a suitably chosen template.

Next, we focus on the complexity of subsets of counting quantifiers on clique and cycle templates. For cycles we give a full trichotomy – all such problems are in L, NP-complete or Pspace-complete. For cliques we come close to a similar trichotomy, but one class remains outstanding.

Afterwards, we consider the generalisation of CSPs in which we augment the extant quantifier $\exists^{\geq 1} := \exists$ with the quantifier $\exists^{\geq j}$ $(j \neq 1)$. Such a CSP is already NP-hard on non-bipartite graph templates. We explore the situation of this generalised CSP on bipartite templates, giving various conditions for both tractability and hardness – culminating in a classification theorem for general graphs.

Finally, we use counting quantifiers to solve the complexity of a concrete QCSP whose complexity was previously open.

1 Introduction

The *constraint satisfaction problem* $\mathrm{CSP}(\mathcal{B})$, much studied in artificial intelligence, is known to admit several equivalent formulations, two of the best known of which are the query evaluation of primitive positive (pp) sentences – those involving only existential quantification and conjunction – on $\mathcal{B}$, and the homomorphism problem to $\mathcal{B}$ (see, e.g., [18]). The problem $\mathrm{CSP}(\mathcal{B})$ is NP-complete in general, and a great deal of effort has been expended in classifying its complexity

* The authors gratefully acknowledge the facilities of the Fields Institute in Toronto where the some of the work on this project was done during the Thematic Program of the institute in Mathematics of Constraint Satisfaction in August 2011.
** Author supported by EPSRC grant EP/G020604/1.
*** Author supported by EPSRC grant EP/I01795X/1.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 253–265, 2012.

for certain restricted cases. Notably it is conjectured [15,6] that for all fixed $\mathcal{B}$, the problem CSP($\mathcal{B}$) is in P or NP-complete. While this has not been settled in general, a number of partial results are known – e.g. over structures of size at most three [25,5] and over smooth digraphs [16,1].

A popular generalisation of the CSP involves considering the query evaluation problem for *positive Horn* logic – involving only the two quantifiers, $\exists$ and $\forall$, together with conjunction. The resulting *quantified constraint satisfaction problems* QCSP($\mathcal{B}$) allow for a broader class, used in artificial intelligence to capture non-monotonic reasoning, whose complexities rise to Pspace-completeness.

In this paper, we study counting quantifiers of the form $\exists^{\geq j}$, which allow one to assert the existence of at least j elements such that the ensuing property holds. Thus on a structure $\mathcal{B}$ with domain of size n, the quantifiers $\exists^{\geq 1}$ and $\exists^{\geq n}$ are precisely $\exists$ and $\forall$, respectively. Counting quantifiers have been extensively studied in finite model theory (see [11,22]), where the focus is on supplementing the descriptive power of various logics. Of more general interest is the majority quantifier $\exists^{\geq n/2}$ (on a structure of domain size n), which sits broadly midway between $\exists$ and $\forall$. Majority quantifiers are studied across diverse fields of logic and have various practical applications, e.g. in cognitive appraisal and voting theory [10]. They have also been studied in computational complexity, e.g., in [19].

We study variants of CSP($\mathcal{B}$) in which the input sentence to be evaluated on $\mathcal{B}$ (of size $|B|$) remains positive conjunctive in its quantifier-free part, but is quantified by various counting quantifiers.

For $X \subseteq \{1, \ldots, |B|\}$, $X \neq \emptyset$, the X-CSP($\mathcal{B}$) takes as input a sentence given by a conjunction of atoms quantified by quantifiers of the form $\exists^{\geq j}$ for $j \in X$. It then asks whether this sentence is true on $\mathcal{B}$. The idea to study $\{1, \ldots, |B|\}$-CSP($\mathcal{B}$) is originally due to Andrei Krokhin.

In Section 3, we consider the power of a single quantifier $\exists^{\geq j}$. We prove that for each $n \geq 3$, there is a template $\mathcal{B}_n$ of size n, such that $\exists^{\geq j}$ $(1 < j < n)$ already has the full complexity of QCSP, i.e., $\{j\}$-CSP($\mathcal{B}_n$) is Pspace-complete.

In Section 4, we go on to study the complexity of subsets of our quantifiers on clique and cycle templates, $\mathcal{K}_n$ and $\mathcal{C}_n$, respectively. We derive the following classification theorems.

Theorem 1. *For $n \in \mathbb{N}$ and $X \subseteq \{1, \ldots, n\}$:*

(i) *X-CSP($\mathcal{K}_n$) is in L if $n \leq 2$ or $X \cap \{1, \ldots, \lfloor n/2 \rfloor\} = \emptyset$.*
(ii) *X-CSP($\mathcal{K}_n$) is NP-complete if $n > 2$ and $X = \{1\}$.*
(iii) *X-CSP($\mathcal{K}_n$) is Pspace-complete if $n > 2$ and either $j \in X$ for $1 < j < n/2$ or $\{1, j\} \subseteq X$ for $j \in \{\lceil n/2 \rceil, \ldots, n\}$.*

This is a near trichotomy – only the cases where n is even and we have the quantifier $\exists^{\geq n/2}$ remain open. For cycles, however, the trichotomy is complete.

Theorem 2. *For $n \geq 3$ and $X \subseteq \{1, \ldots, n\}$, the problem X-CSP($\mathcal{C}_n$) is either in L, is NP-complete or is Pspace-complete. Namely:*

(i) *X-CSP($\mathcal{C}_n$) $\in$ L if $n = 4$, or $1 \notin X$, or n is even and $X \cap \{2, \ldots, n/2\} = \emptyset$.*
(ii) *X-CSP($\mathcal{C}_n$) is NP-complete if n is odd and $X = \{1\}$.*
(iii) *X-CSP($\mathcal{C}_n$) is Pspace-complete in all other cases.*

In Section 5, we consider $\{1, j\}$-CSP($\mathcal{H}$), for $j \neq 1$ on graphs. The CSP is already NP-hard on non-bipartite graph templates. We explore the situation of this generalised CSP on bipartite graph templates, giving various conditions for both tractability and hardness, using and extending results of Section 4. We are most interested here in the distinction between P and NP-hard. To understand which of these cases are Pspace-complete would include as a subclassification the Pspace-complete cases of QCSP($\mathcal{H}$), a question which has remained open for five years [21]. We give a classification theorem for graphs in fragments of the logic involving bounded use of $\exists^{\geq 2}$ followed by unbounded use of $\exists$. In the case of QCSP ($\exists^{\geq n}$ instead of $\exists^{\geq 2}$), this is perfectly natural and is explored with bounded alternations in, e.g., [8,9,17], and with bounded use of $\forall = \exists^{\geq n}$ in [7]. We prove that either there exists such a fragment in which the problem is NP-hard or for all such fragments the problem is in P.

Afterwards in Section 6, we use counting quantifiers to solve the complexity of QCSP($\mathcal{C}_4^*$), where $\mathcal{C}_4^*$ is the reflexive 4-cycle, whose complexity was previously open. Finally, in Section 7, we give some closing remarks and open problems.

2 Preliminaries

Let $\mathcal{B}$ be a finite structure over a finite signature σ whose domain B is of cardinality $|B|$. For $1 \leq j \leq |B|$, the formula $\exists^{\geq j} x\ \phi(x)$ with *counting quantifier* should be interpreted on $\mathcal{B}$ as stating that there exist at least j distinct elements $b \in B$ such that $\mathcal{B} \models \phi(b)$. Counting quantifiers generalise existential ($\exists := \exists^{\geq 1}$), universal ($\forall := \exists^{\geq |B|}$) and (weak) majority ($\exists^{\geq |B|/2}$) quantifiers. Counting quantifiers do not in general commute with themselves, viz $\exists^{\geq j} x \exists^{\geq j} y \neq \exists^{\geq j} y \exists^{\geq j} x$ (in contrast, $\exists$ and $\forall$ do commute with themselves, but not with one another).

For $\emptyset \neq X \subseteq \{1, \ldots, |B|\}$, the X-CSP($\mathcal{B}$) takes as input a sentence of the form $\Phi := Q_1 x_1 Q_2 x_2 \ldots Q_m x_m\ \phi(x_1, x_2, \ldots, x_m)$, where ϕ is a conjunction of positive atoms of σ and each Q_i is of the form $\exists^{\geq j}$ for some $j \in X$. The set of such sentences forms the logic X-pp (recall the pp is primitive positive). The yes-instances are those for which $\mathcal{B} \models \Phi$. Note that all problems X-CSP($\mathcal{B}$) are trivially in Pspace, by cycling through all possible evaluations for the variables.

The problem $\{1\}$-CSP($\mathcal{B}$) is better-known as just CSP($\mathcal{B}$), and $\{1, |B|\}$-CSP($\mathcal{B}$) is better-known as QCSP($\mathcal{B}$). We will consider also the logic $[2^m 1^*]$-pp and restricted problem $[2^m 1^*]$-CSP($\mathcal{B}$), in which the input $\{1, 2\}$-pp sentence has prefix consisting of no more than m $\exists^{\geq 2}$ quantifiers followed by any number of $\exists$ quantifiers (and nothing else).

A homomorphism from $\mathcal{A}$ to $\mathcal{B}$, both σ-structures, is a function $h : A \to B$ such that $(a_1, \ldots, a_r) \in R^{\mathcal{A}}$ implies $(h(a_1), \ldots, h(a_r)) \in R^{\mathcal{B}}$, for all relations R of σ. A frequent role will be played by the *retraction* problem Ret($\mathcal{B}$) in which one is given a structure $\mathcal{A}$ containing $\mathcal{B}$, and one is asked if there is a homomorphism from $\mathcal{A}$ to $\mathcal{A}$ that is the identity on $\mathcal{B}$. It is well-known that retraction problems are special instances of CSPs in which the constants of the template are all named [12].

In line with convention we consider the notion of hardness reduction in proofs to be polynomial many-to-one (though logspace is sufficient for our results).

2.1 Game Characterisation

There is a simple game characterisation for the truth of sentences of the logic X-pp on a structure $\mathcal{B}$. Given a sentence Ψ of X-pp, and a structure $\mathcal{B}$, we define the following game $\mathscr{G}(\Psi, \mathcal{B})$. Let $\Psi := Q_1 x_1 Q_2 x_2 \dots Q_m x_m \ \psi(x_1, x_2, \dots, x_m)$. Working from the outside in, coming to a quantified variable $\exists^{\geq j} x$, the *Prover* (female) picks a subset B_x of j elements of B as witnesses for x, and an *Adversary* (male) chooses one of these, say b_x, to be the value of x. Prover wins iff $\mathcal{B} \models \psi(b_{x_1}, b_{x_2}, \dots, b_{x_m})$. The following comes immediately from the definitions.

Lemma 1. *Prover has a winning strategy in the game $\mathscr{G}(\Psi, \mathcal{B})$ iff $\mathcal{B} \models \Psi$.*

We will often move seemlessly between the two characterisations of Lemma 1. One may alternatively view the game in the language of homomorphisms. There is an obvious bijection between σ-structures with domain $\{1, \dots, m\}$ and conjunctions of positive atoms in variables $\{v_1, \dots, v_m\}$. From a structure $\mathcal{B}$ build the conjunction $\phi_{\mathcal{B}}$ listing the tuples that hold on $\mathcal{B}$ in which element i corresponds to variable v_i. Likewise, for a conjunction of positive atoms ψ, let $\mathcal{D}_\psi$ be the structure whose relation tuples are listed by ψ, where variable v_i corresponds to element i. The relationship of $\mathcal{B}$ to $\phi_{\mathcal{B}}$ and ψ to $\mathcal{D}_\psi$ is very similar to that of *canonical query* and *canonical database* (see [18]), except there we consider the conjunctions of atoms to be existentially quantified. For example, $\mathcal{K}_3$ on domain $\{1, 2, 3\}$ gives rise to $\phi_{\mathcal{K}_3} := \exists v_1, v_2, v_3 \ E(v_1, v_2) \wedge E(v_2, v_1) \wedge E(v_2, v_3) \wedge E(v_3, v_2) \wedge E(v_3, v_1) \wedge E(v_3, v_1)$. The Prover-Adversary game $\mathscr{G}(\Psi, \mathcal{B})$ may be seen as Prover giving j potential maps for element x in $\mathcal{D}_\psi$ (ψ is quantifier-free part of Ψ) and Adversary choosing one of them. The winning condition for Prover is now that the map given from $\mathcal{D}_\psi$ to $\mathcal{B}$ is a homomorphism.

In the case of QCSP, i.e. $\{1, |B|\}$-pp, each move of a game $\mathscr{G}(\Psi, \mathcal{B})$ is trivial for one of the players. For $\exists^{\geq 1}$ quantifiers, Prover gives a singleton set, so Adversary's choice is forced. In the case of $\exists^{\geq |B|}$ quantifiers, Prover must advance all of B. Thus, essentially, Prover alone plays $\exists^{\geq 1}$ quantifiers and Adversary alone plays $\exists^{\geq |B|}$ quantifiers.

3 Complexity of a Single Quantifier

In this section we consider the complexity of evaluating X-pp sentences when X is a singleton, i.e., we have at our disposal only a single quantifier.

Theorem 3.
(1) $\{1\}$*-CSP*$(\mathcal{B})$ *(i.e. CSP*$(\mathcal{B})$*) is in* NP *for all* $\mathcal{B}$*. For each* $n \geq 2$*, there exists a template* $\mathcal{B}_n$ *of size* n *such that* $\{1\}$*-CSP*$(\mathcal{B}_n)$ *is* NP*-complete.*
(2) $\{|B|\}$*-CSP*$(\mathcal{B})$ *is in* L *for all* $\mathcal{B}$*.*
(3) For each $n \geq 3$*, there exists a template* $\mathcal{B}_n$ *of size* n *such that* $\{j\}$*-CSP*$(\mathcal{B}_n)$ *is* Pspace*-complete for all* $1 < j < n$*.*

Proof. Parts (1) and (2) are well-known (see [23], resp. [20]). For (3), let $\mathcal{B}_{\mathrm{NAE}}$ be the Boolean structure on domain $\{0, 1\}$ with a single ternary not-all-equal relation $R_{\mathrm{NAE}} := \{0, 1\}^3 \setminus \{(0, 0, 0), (1, 1, 1)\}$. To show Pspace-completeness, we reduce from QCSP$(\mathcal{B}_{\mathrm{NAE}})$, the *quantified not-all-equal-*3*-satisfiability* (see [23]).

We distinguish two cases.

Case I: $j \leq \lfloor n/2 \rfloor$. Define $\mathcal{B}_n$ on domain $\{0, \ldots, n-1\}$ with a single unary relation U and a single ternary relation R. Set $U := \{0, \ldots, j-1\}$ and set

$$R := \{0, \ldots, n-1\}^3 \setminus \{(a,b,c) : \ a, b, c \text{ either all odd or all even}\}.$$

The even numbers will play the role of false 0 and odd numbers the role of true 1.

Case II: $j > \lfloor n/2 \rfloor$. Define $\mathcal{B}_n$ on domain $\{0, \ldots, n-1\}$ with a single unary relation U and a single ternary relation R. Set $U := \{0, \ldots, j-1\}$ and set

$$R := \{0, \ldots, n-1\}^3 \setminus \{(a,b,c) : \ a, b, c \leq n-j \text{ and either all odd or all even}\}.$$

In this case even numbers $\leq n-j$ play the role of false 0 and odd numbers $\leq n-j$ play the role of true 1. The $j-1$ numbers $n-j+1, \ldots, n-1$ are somehow universal and will always satisfy any R relation.

The reduction we use is the same for Cases I and II. We reduce QCSP($\mathcal{B}_{\text{NAE}}$) to $\{j\}$-CSP($\mathcal{B}_n$). Given an input $\Psi := Q_1 x_1 Q_2 x_2 \ldots Q_m x_m \ \psi(x_1, x_2, \ldots, x_m)$ to the former (i.e. each Q_i is $\exists$ or $\forall$) we build an instance Ψ' for the latter. From the outside in, we convert quantifiers $\exists x$ to $\exists^{\geq j} x$. For quantifiers $\forall x$, we convert also to $\exists^{\geq j} x$, but we add the conjunct $U(x)$ to the quantifier-free part ψ.

We claim $\mathcal{B}_{\text{NAE}} \models \Psi$ iff $\mathcal{B}_n \models \Psi'$. For the $\exists$ variables of Ψ, we can see that any j witnesses from the domain B_n for $\exists^{\geq j}$ must include some element playing the role of either false 0 or true 1 (and the other $j-1$ may always be found somewhere). For the $\forall$ variables of Ψ, U forces us to choose both 0 and 1 among the $\exists^{\geq j}$ (and the other $j-2$ will come from $2, \ldots, j-1$). The result follows. □

4 Counting Quantifiers on Cliques and Cycles

4.1 Cliques: Proof of Theorem 1

Recall that $\mathcal{K}_n$ is the complete irreflexive graph on n vertices.

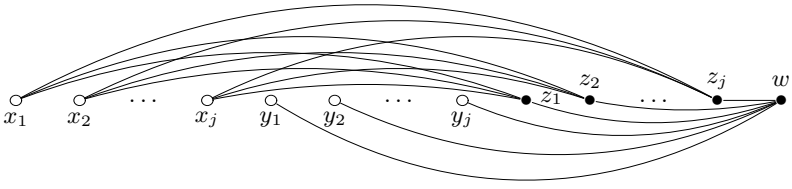


Fig. 1. The gadget $\mathcal{G}_j$

Proposition 1. *If $1 < j$, then $\{j\}$-CSP($\mathcal{K}_{2j+1}$) is* Pspace*-complete.*

Proof. By reduction from QCSP$\left(\mathcal{K}_{\binom{2j+1}{j}}\right)$, *quantified* $\binom{2j+1}{j}$*-colouring*, which is Pspace-complete by [4]. The key part of our proof involves the gadget $\mathcal{G}_j$, in Figure 1, having vertices $x_1, \ldots, x_j, y_1, \ldots, y_j, z_1, \ldots, z_j, w$ and all possible edges between $\{x_1, \ldots, x_j\}$ and $\{z_1, \ldots, z_j\}$, and between w and $\{y_1, \ldots, y_j, z_1, \ldots, z_j\}$. The left $2j$ vertices represent free variables $x_1, \ldots, x_j$, $y_1, \ldots, y_j$. Observe that $\exists^{\geq j} z_1, \ldots, z_j, w \ \phi_{\mathcal{G}_j}$ is true on $\mathcal{K}_{2j+1}$ iff $|\{x_1, \ldots, x_j\} \cap \{y_1, \ldots, y_j\}| < j$. If

$|\{x_1,\ldots,x_j\}| = |\{y_1,\ldots,y_j\}| = j$, this is equivalent to $\{x_1 \ldots x_j\} \neq \{y_1 \ldots y_j\}$. Thus this gadget will help us to encode the edge relation on $\mathcal{K}_{\binom{2j+1}{j}}$ where we represent vertices by sets $\{a_1,\ldots,a_j\} \subset \{1,\ldots,2j+1\}$ with $|\{a_1,\ldots,a_j\}| = j$.

Consider an instance Ψ of $\mathrm{QCSP}\big(\mathcal{K}_{\binom{2j+1}{j}}\big)$. We construct the instance Ψ' of $\{j\}$-CSP($\mathcal{K}_{2j+1}$) as follows. From the graph $\mathcal{D}_\psi$, build $\mathcal{D}_{\psi'}$ by transforming each vertex v into an independent set of j vertices $\{v^1,\ldots,v^j\}$, and each edge uv of $\mathcal{D}_\psi$ to an instance of the gadget $\mathcal{G}_j$ in which the $2j$ free variables correspond to $u^1,\ldots,u^j,v^1,\ldots,v^j$. The other variables of the gadget $\{z_1,\ldots,z_j,w\}$ are unique to each edge and are quantified innermost in Ψ' in the order $z_1,\ldots,z_j,w$.

It remains to explain the quantification of the variables of the form $v^1,\ldots,v^j$. We follow the quantifier order of Ψ. Existentially quantified variables $\exists v$ of Ψ are quantified as $\exists^{\geq j} v^1,\ldots,v^j$ in Ψ'. Universally quantified variables $\forall v$ of Ψ are also quantified $\exists^{\geq j} v^1,\ldots,v^j$ in Ψ', but we introduce additional variables $v^{1,1},\ldots,v^{1,j+1},\ \ldots,\ v^{j,1},\ldots,v^{j,j+1}$ before $v^1,\ldots,v^j$ in the quantifier order of Ψ', and for each $i \in \{1,\ldots,j\}$, we join $v^{i,1},\ldots,v^{i,j+1}$ into a clique with v^i.

It is now not difficult to verify that $\mathcal{K}_{\binom{2j+1}{j}} \models \Psi$ iff $\mathcal{K}_{2j+1} \models \Psi'$. □

Corollary 1. *If $1 < j < n/2$, then $\{j$-CSP($\mathcal{K}_n)$ is* Pspace*-complete.*

Proof. We reduce from $\{j\}$-CSP($\mathcal{K}_{2j+1}$) and appeal to Proposition 1. Given an input Ψ for $\{j\}$-CSP($\mathcal{K}_{2j+1}$), we build an instance Ψ' for $\{j\}$-CSP($\mathcal{K}_n$) by adding an $(n-2j-1)$-clique on new variables, quantified outermost in Ψ', and link by an edge each variable of this clique to every other variable. Adversary chooses $n-2j-1$ elements of the domain for this clique, effectively reducing the domain size to $2j+1$ for the rest. Thus $\mathcal{K}_n \models \Psi'$ iff $\mathcal{K}_{2j+1} \models \Psi$ follows. □

Proposition 2. *If $1 < j \leq n$, then $\{1,j$-CSP($\mathcal{K}_n)$ is* Pspace*-complete.*

Proof. By reduction from QCSP($\mathcal{K}_n$). We simulate existential quantification $\exists v$ by itself, and universal quantification $\forall v$ by the introduction of $(n-j+1)$ new variables $v^1,\ldots,v^{n-j}$, joined in a clique with v, and quantified by $\exists^{\geq j}$ before v (which is also quantified by $\exists^{\geq j}$). The argument follows as in Proposition 1. □

Define the n-star $\mathcal{K}_{1,n}$ to be the graph on vertices $\{0,1,\ldots,n\}$ with edges $\{(0,j),(j,0)\ :\ j \geq 1\}$ where 0 is called the *centre* and the remainder are *leaves.*

Proposition 3. *If $X \cap \{1,\ldots,\lfloor n/2 \rfloor\} = \emptyset$, then X-CSP($\mathcal{K}_n$) is in* L.

Proof. Instance Ψ of X-CSP($\mathcal{K}_n$) of the form $\exists^{\geq \lambda_1} x_1 \ldots \exists^{\geq \lambda_m} x_m\ \psi(x_1,\ldots,x_m)$ induces the graph $\mathcal{D}_\psi$, which we may consider totally ordered (the order is given left-to-right ascending by the quantifiers). We claim that $\mathcal{K}_n \models \Psi$ iff $\mathcal{D}_\psi$ does not contain as a subgraph (not necessarily induced) a $(n-\lambda_i+1)$-star in which the $n-\lambda_i+1$ leaves all come before the centre x_i in the ordering.

($\Rightarrow$) If $\mathcal{D}_\psi$ contains such a star, then Ψ is a no-instance, as we may give a winning strategy for Adversary in the game $\mathcal{G}(\Psi,\mathcal{K}_n)$. Adversary should choose distinct values for the variables associated with the $n-\lambda_i+1$ leaves of the star (can always be done as each of the possible quantifiers assert existence of $> n/2$ elements and $n-\lambda_i < n/2$), whereupon there is no possibility for Prover to choose λ_i witnesses to the variable x_i associated with the centre.

($\Leftarrow$) If $\mathcal{D}_\psi$ does not contain such a star, then we give the following winning strategy for Prover in the game $\mathscr{G}(\Psi, \mathcal{K}_n)$. Whenever a new variable comes up, its corresponding vertex in $\mathcal{D}_\psi$ has $l < n - \lambda_i + 1$ adjacent predecessors, which were answered with $b_1, \ldots, b_l$. Prover suggests any set of size λ_i from $B \setminus \{b_1, \ldots, b_l\}$ (which always exists) and the result follows. □

Proof of Theorem 1. For $n \leq 2$ see [21], and for (ii) see [16]. The remainder of (i) is proved as Proposition 3 while Corollary 1 and Proposition 2 give (iii). □

4.2 Cycles: Proof of Theorem 2

Recall that $\mathcal{C}_n$ denotes the irreflexive symmetric cycle on n vertices. We consider $\mathcal{C}_n$ to have vertices $\{0, 1, \ldots, n-1\}$ and edges $\big\{(i,j) \;:\; |i-j| \in \{1, n-1\}\big\}$.

In the forthcoming proof, we use the following elementary observation from additive combinatorics. Let $n \geq 2$, $j \geq 1$, and A, B be sets of integers. Define:

- $A +_n B = \big\{(a+b) \bmod n \;\big|\; a \in A,\, b \in B\big\}$
- $j \times_n A = \underbrace{A +_n \ldots +_n A}_{j \text{ times}}$

Lemma 2. *Let $n \geq 3$ and $2 \leq j < n$. Then*

$$\Big|\, j \times_n \{-1, +1\} \,\Big| = \begin{cases} j+1 & n \text{ is odd} \\ \min\{j+1, n/2\} & n \text{ is even} \end{cases}$$

$$\Big|\, n \times_n \{-1, +1\} \,\Big| = \Big|\, n \times_n \{-2, 0, +2\} \,\Big| = \begin{cases} n & n \text{ is odd} \\ n/2 & n \text{ is even} \end{cases}$$

Proposition 4. *If $n \geq 3$, then X-CSP($\mathcal{C}_n$) is in* L *if $n = 4$, or $1 \notin X$, or n is even and $X \cap \{2, 3 \ldots, n/2\} = \emptyset$,*

Proof. Let Ψ be an instance of X-CSP($\mathcal{C}_n$). Recall that $\mathcal{D}_\psi$ is the graph corresponding to the quantifier-free part of Ψ. We write $x \prec y$ if x, y are vertices of $\mathcal{D}_\psi$ (i.e., variables of ψ) such that x is quantified before y in Ψ. For an edge xy of $\mathcal{D}_\psi$ where $x \prec y$, we say that x is a *predecessor* of y. Note that a vertex can have several predecessors. The following restricts the yes-instances of X-CSP($\mathcal{C}_n$).

Let x be a vertex of $\mathcal{D}_\psi$ quantified in Ψ by $\exists^{\geq j}$ for some j. If $\mathcal{C}_n \models \Psi$ then

(1a) if $j \geq 3$, then x has no predecessors,

(1b) if n is even and $j > n/2$, then x is the first vertex (w.r.t. $\prec$) of some connected component of $\mathcal{D}_\psi$, and

(1c) if $n \neq 4$ and $j = 2$, then all predecessors of x except for its first predecessor (w.r.t. $\prec$) are quantified by $\exists^{\geq 1}$.

Using these we prove the proposition. First, we consider the case $n = 4$. We show that $\{1, 2, 3, 4\}$-CSP($\mathcal{C}_4$) is in L. This will imply that X-CSP($\mathcal{C}_4$) is in L for every X. Observe that if $\mathcal{D}_\psi$ contains a vertex x quantified by $\exists^{\geq 3}$ or $\exists^{\geq 4}$, then by (1b) this vertex is the first in its component (if Ψ is not a trivial no-instance). Thus by symmetry replacing its quantification by $\exists^{\geq 1}$ does not change the truth of Ψ. So we may assume that Ψ is an instance of $\{1, 2\}$-CSP($\mathcal{C}_4$). We now claim that $\mathcal{C}_4 \models \Psi$ if and only if $\mathcal{D}_\psi$ is bipartite. Clearly, if $\mathcal{D}_\psi$ is not bipartite, it has no homomorphism to $\mathcal{C}_4$ and hence $\mathcal{C}_4 \not\models \Psi$. Conversely, assume that $\mathcal{D}_\psi$ is bipartite with bipartition (A, B). Our strategy for Prover offers the set $\{0, 2\}$ or its subsets for the vertices in A and offers $\{1, 3\}$ or its subsets for every vertex in B. It is easy

to verify that this is a winning strategy for Prover. Thus $\mathcal{C}_4 \models \Psi$. The complexity now follows as checking (1b) and checking if a graph is bipartite is in L by [24].

Now, we may assume $n \neq 4$, and next we consider the case $1 \notin X$. If also $2 \notin X$, then by (1a) the graph $\mathcal{D}_\psi$ contains no edges (otherwise Ψ is a trivial no-instance). This is clearly easy to check in L. Thus $2 \in X$. We claim that if we satisfy (1a) and (1c), then $\mathcal{C}_n \models \Psi$. We provide a winning strategy for Prover. Namely, for a vertex x, if x has no predecessors, offer any set for x. If x has a unique predecessor y for which the value i was chosen, then x is quantified by $\exists^{\geq 2}$ (or $\exists$) by (1a) and we offer $\{i-1, i+1\} \pmod n$ for x . There are no other cases by (1a) and (1c). It follows that Prover always wins with this strategy. In terms of complexity, it suffices to check (1a) and (1c) which is in L.

Finally, suppose that n is even and $X \cap \{2 \dots n/2\} = \emptyset$. Note that every vertex of $\mathcal{D}_\psi$ is either quantified by $\exists^{\geq 1}$ or by $\exists^{\geq j}$ where $j > n/2$. Thus, using (1b), unless Ψ is a trivial no-instance, we can again replace every $\exists^{\geq j}$ in Ψ by $\exists^{\geq 1}$ without changing the truth of Ψ. Hence, we may assume that Ψ is an instance of $\{1\}$-CSP$(\mathcal{C}_n)$. Thus, as n is even, $\mathcal{C}_n \models \Psi$ if and only if $\mathcal{D}_\psi$ is bipartite. The complexity again follows from [24]. That concludes the proof. □

Proposition 5. *Let $n \geq 3$. Then X-CSP$(\mathcal{C}_n)$ is Pspace-complete if $n \neq 4$ and $\{1, j\} \subseteq X$: where $j \in \{2, \dots, n\}$ if n is odd and $j \in \{2, \dots, n/2\}$ if n is even.*

Proof. By reduction, namely a reduction from QCSP$(\mathcal{C}_n)$ for odd n, and from QCSP$(\mathcal{K}_{n/2})$ for even n. Both problems are known to be Pspace-hard [4].

First, consider the case of odd n. Let Ψ be an instance of QCSP$(\mathcal{C}_n)$. In other words, Ψ is an instance of $\{1, n\}$-CSP$(\mathcal{C}_n)$. Clearly, $j < n$ otherwise we are done.

We modify Ψ by replacing each universally-quantified variable x of Ψ by a path. Namely, let π_x denote the pp-formula that encodes that

$$x_1^1, x_2^1, \dots, x_{j-1}^1,\ x_1^2, x_2^2, \dots, x_{j-1}^2, \quad \dots \quad , x_1^n, x_2^n, \dots, x_{j-1}^n,\ x$$

is a path in that order (all but x are new variables). We replace $\forall x$ by

$$Q_x \ = \ \exists^{\geq j} x_1^1\ \exists^{\geq j} x_1^2 \dots\ \exists^{\geq j} x_1^n\ \exists^{\geq j} x\ \exists^{\geq 1} x_2^1 \dots \exists^{\geq 1} x_{j-1}^1\ \dots\ \exists^{\geq 1} x_2^n \dots \exists^{\geq 1} x_{j-1}^n$$

and append π_x to the quantifier-free part of the formula. Let Ψ' denote the final formula after considering all universally quantified variables. Note that Ψ' is an instance of $\{1, j\}$-CSP$(\mathcal{C}_n)$. We claim that $\mathcal{C}_n \models \Psi$ if and only if $\mathcal{C}_n \models \Psi'$.

To do this, it suffices to show that Ψ' correctly simulates the universal quantifiers of Ψ. Namely, that $\mathcal{C}_n \models Q_x \pi_x$, and for each $\ell \in \{0 \dots n-1\}$, Adversary has a strategy on $Q_x \pi_x$ that evaluates x to ℓ. (We omit further details.)

It remains to investigate the case of even n. Recall that $n \geq 6$ and $j \leq n/2$. We show a reduction from QCSP$(\mathcal{K}_{n/2})$ to $\{1, j\}$-QCSP$(\mathcal{C}_n)$. The reduction is a variant of the construction from [13] for the problem of retraction to even cycles.

Let Ψ be an instance of QCSP$(\mathcal{K}_{n/2})$, and define $r = (-n/2 - 2) \bmod (j-1)$. We construct a formula Ψ' from Ψ as follows. First, we modify Ψ by replacing universal quantifiers exactly as in the case of odd n. Namely, we define Q_x and π_x as before, replace each $\forall x$ by Q_x, and append π_x to the quantifier-free part of the formula. After this, we append to the formula a cycle on n vertices $v_0, v_1, \dots, v_{n-1}$ with a path on $r+1$ vertices $w_0, w_1, \dots, w_r$. (See the black

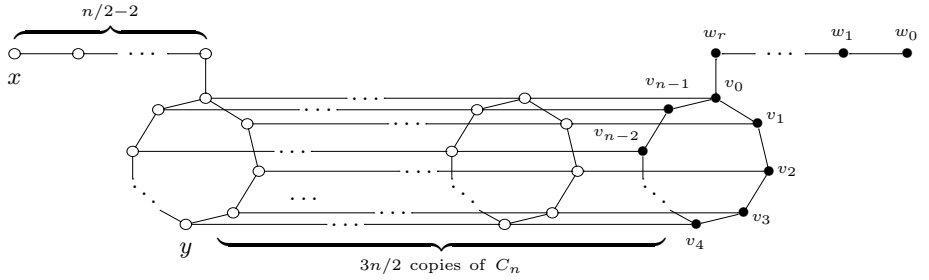


Fig. 2. The gadget for the case of even n where $r = (-n/2 - 2) \bmod (j - 1)$

vertices in Figure 2.) Then, for each edge xy of $\mathcal{D}_\psi$, we replace $E(x, y)$ in Ψ by the gadget depicted in Figure 2 (consisting of the cartesian product of $\mathcal{C}_n$ and a path on $3n/2$ vertices together with two attached paths on $n/2 - 2$, resp. $r + 1$ vertices). The vertices x and y represent the variables x and y while all other white vertices are new variables, and the black vertices are identified with $v_0, \ldots, v_{n-1}, w_0, \ldots, w_r$ introduced in the previous step.

Finally, we prepend the following quantification to the formula:

$$\exists^{\geq 1} w_0 \, \exists^{\geq j} v_{j-r-2} \, \exists^{\geq j} v_{2j-r-3} \ldots \exists^{\geq j} v_{(k \cdot j - r - k - 1)} \ldots \exists^{\geq j} v_{n/2+1}$$

followed by $\exists^{\geq 1}$ quantification of all the remaining variables of the gadgets.

We prove that $\mathcal{K}_{n/2} \models \Psi$ if and only if $\mathcal{C}_n \models \Psi'$. First, we show that Ψ' correctly simulates the universal quantification of Ψ. The argument for this is essentially the same as in the case of odd n. Next, we need to analyse possible assignments to the vertices $v_0, \ldots, v_{n-1}$. There are two cases: either the values chosen for $v_0, \ldots, v_{n-1}$ are all distinct, or not. In the former, we show that Prover can complete the homomorphism to $\mathcal{C}_n$ if and only if $\mathcal{K}_{n/2} \models \Psi$. All other cases are degenerate and have to be addressed separately. (We omit the details.) □

Proof of Theorem 2. The case (i) is proved as Proposition 4, and the case (ii) follows from [16]. Finally, the case (iii) is proved as Proposition 5. □

5 Extensions of the CSP

In this section we consider single-quantifier extensions of the classical CSP($\mathcal{B}$), i.e., the evaluation of X-pp sentences, where $X := \{1, j\}$ for some $1 < j \leq |B|$.

5.1 Bipartite Graphs

In the case of (irreflexive, undirected) graphs, it is known that $\{1\}$-CSP($\mathcal{H}$) = CSP($\mathcal{H}$) is in L if $\mathcal{H}$ is bipartite and is NP-complete otherwise [16] (for membership in L, one needs also [24]). It is also known that something similar holds for $\{1, |H|\}$-CSP($\mathcal{H}$) = QCSP($\mathcal{H}$) – this problem is in L if $\mathcal{H}$ is bipartite and is NP-hard otherwise [21]. Of course, the fact that $\{1, j\}$-CSP($\mathcal{H}$) is hard on non-bipartite $\mathcal{H}$ is clear, but we will see that it is not always easy on bipartite $\mathcal{H}$.

First, we look at complete bipartite graphs (in a more general statement).

Proposition 6. *Let $\mathcal{K}_{k,l}$ be the complete bipartite graph with partite sets of size k and l. Then $\{1,\ldots,k+l\}$-CSP($\mathcal{K}_{k,l}$) is in* L.

Proof. We reduce to QCSP($\mathcal{K}_2^1$), where $\mathcal{K}_2^1$ indicates $\mathcal{K}_2$ with one vertex named by a constant, say 1. QCSP($\mathcal{K}_2^1$) is equivalent to QCSP($\mathcal{K}_2$) (identify instances of 1 to a single vertex) and both are well-known to be in L (see, e.g., [21]). Let Ψ be input to $\{1,\ldots,k+l\}$-CSP($\mathcal{K}_{k,l}$). Produce Ψ' by substituting quantifiers $\exists^{\geq j}$ with $\exists$, if $j \leq \min\{k,l\}$, or with $\forall$, if $j > \max\{k,l\}$. Variables quantified by $\exists^{\geq j}$ for $\min\{k,l\} < j \leq \max\{k,l\}$ should be replaced by the constant 1. It is easy to see that $\mathcal{K}_{k,l} \models \Psi$ iff $\mathcal{K}_2 \models \Psi'$, and the result follows. □

Proposition 7. *For each j, there exists m s.t. $[2^m1^*]$-CSP($\mathcal{C}_{2j}$) is* NP*-complete.*

Proof. Membership in NP follows because m is bounded – one may try all possible evaluations to the $\exists^{\geq 2}$ variables. NP-hardness follows as in the proof of case (*iii*) of Theorem 2, but we are reducing from CSP($\mathcal{K}_{n/2}$) not QCSP($\mathcal{K}_{n/2}$). As a consequence, the only instances of $\exists^{\geq 2}$ we need to consider are those used to isolate the cycle $\mathcal{C}_{2j}$ (one may take $m := j+3$). □

Corollary 2. *If $\mathcal{H}$ is bipartite and (for $j \geq 3$) contains some $\mathcal{C}_{2j}$ but no smaller cycle, then exists m s.t. $[2^m1^*]$-CSP($\mathcal{H}$) is* NP*-complete.*

Proof. Membership and reductions for hardness follow similarly to Proposition 7. The key part is in isolating a copy of the cycle, but we can not do this as easily as before. If d is the diameter of $\mathcal{H}$ (the maximum of the minimal distances between two vertices) then we begin the sentence Ψ' of the reduction with $\exists^{\geq 2}v_1,\ldots,v_{d+1}$, and then, for each $i \in \{1,\ldots,d-j+1\}$ we add $\exists^{\geq 2}x_i, x_i', \ldots, x'^{\cdots\prime}$ ($j-1$ dashes) and join $v_i,\ldots,v_{i+j},x_i'^{\cdots\prime},\ldots,x_i$ in a $2i$-cycle (with $E(x_i,v_i)$ also). For each of these $d-j+1$ cycles $\mathcal{C}_{2j}$ we build a separate copy of the rest of the reduction. We can not be sure which of these cycles is evaluated truly on some $\mathcal{C}_{2j}$, but at least one of them must be. □

In passing, we note the following simple propositions.

Proposition 8. *If $j \in \{2,...,n-3\}$ then one may exhibit a bipartite $\mathcal{H}_j$ of size n such that $\{1,j\}$-CSP($\mathcal{H}_j$) is* Pspace*-complete.*

Proof. The case $j=2$ follows from Theorem 2; assume $j \geq 3$. Take the graph $\mathcal{C}_6$ and construct $\mathcal{H}_j$ as follows. Augment $\mathcal{C}_6$ with $j-3$ independent vertices each with an edge to vertices 1, 3 and 5 of $\mathcal{C}_6$. Apply the proof of Theorem 2 with $\mathcal{H}_j$. □

Proposition 9. *Let $\mathcal{H}$ be bipartite with largest partition in a connected component of size $< j$. Then $\{1,j\}$-CSP($\mathcal{H}$) is in* L.

Proof. We will consider an input Ψ to $\{1,j\}$-CSP($\mathcal{H}$) of the form $Q_1x_1\ Q_2x_2\ldots Q_mx_m\ \psi(x_1,x_2,\ldots,x_m)$. An instance of an $\exists^{\geq j}$ variable is called *trivial* if it has neither a path to another (distinct) $\exists^{\geq j}$ variable, nor a path to an $\exists$ variable

that precedes it in the natural order on $\mathcal{D}_\psi$. The key observation here is that any non-trivial $\exists^{\geq j}$ variable *must* be evaluated on more than one partition of a connected component. If in Ψ there is a non-trivial $\exists^{\geq j}$ variable, then Ψ must be a no-instance (as $\exists^{\geq j}$s must be evaluated on more than one partition of a connected component, and a path can not be both even and odd in length). All other instances are readily seen to be satisfiable. Detecting if Ψ contains a non-trivial $\exists^{\geq j}$ variable is in L by [24], and the result follows. □

We note that Proposition 8 is tight, namely in that $\{1, j\}$-CSP($\mathcal{H}$) is in L if $j \in \{1, |H| - 2, |H| - 1, |H|\}$. (For space restrictions, we omit the details.)

Proposition 10. *If $\mathcal{H}$ is bipartite and contains $\mathcal{C}_4$, then $\Psi \in \{1, 2\}$-CSP($\mathcal{C}_4$) iff the underlying graph $\mathcal{D}_\psi$ of Ψ is bipartite. In particular, $\{1, 2\}$-CSP($\mathcal{H}$) is in L.*

Proof. Necessity is clear; sufficiency follows by the canonical evaluation of $\exists^{\geq 1}$ and $\exists^{\geq 2}$ on a fixed copy of $\mathcal{C}_4$ in $\mathcal{H}$. Membership in L follows from [24]. □

Proposition 11. *Let $\mathcal{H}$ be a forest, then $[2^m 1^*]$-CSP($\mathcal{H}$) is in* P *for all m.*

Proof. We evaluate each of the m variables bound by $\exists^{\geq 2}$ to all possible pairs, and what we obtain in each case is an instance of CSP($\mathcal{H}'$) where $\mathcal{H}'$ is an expansion of $\mathcal{H}$ by some constants, i.e., equivalent to the retraction problem. It is known that Ret($\mathcal{H}$) is in P for all forests $\mathcal{H}$ [14], and the result follows. □

We bring together some previous results into a classification theorem.

Theorem 4. *Let $\mathcal{H}$ be a graph. Then*

- *$[2^m 1^*]$-CSP($\mathcal{H}$) $\in$ P for all m, if $\mathcal{H}$ is a forest or a bipartite graph containing $\mathcal{C}_4$*
- *$[2^m 1^*]$-CSP($\mathcal{H}$) is NP-complete from some m, if otherwise.*

Proof. Membership of NP follows since m is fixed. The cases in P follow from Propositions 11 and 10. Hardness for non-bipartite graphs follows from [16] and for the remaining bipartite graphs it follows from Corollary 2. □

6 The Complexity of QCSP($\mathcal{C}_4^*$)

Let $\mathcal{C}_4^*$ be the reflexive 4-cycle. The complexities of Ret($\mathcal{C}_6$) and Ret($\mathcal{C}_4^*$) are both hard (NP-complete) [13,12], and retraction is recognised to be a "cousin" of QCSP (see [2]). The problem QCSP($\mathcal{C}_6$) is known to be in L (see [21]), but the complexity of QCSP($\mathcal{C}_4^*$) was hitherto unknown. Perhaps surprisingly, we show that is is markedly different from that of QCSP($\mathcal{C}_6$), being Pspace-complete.

Proposition 12. *$\{1, 2, 3, 4\}$-CSP($\mathcal{C}_4^*$) is* Pspace*-complete.*

Corollary 3. *QCSP($\mathcal{C}_4^*$) is* Pspace*-complete.*

The proofs of these claims are based on the hardness of the retraction problem to reflexive cycles [12] and are similar to our proof of the even case of Proposition 5.

While QCSP($\mathcal{C}_4^*$) has different complexity from QCSP($\mathcal{C}_6$), we remark that the better analog of the retraction complexities is perhaps that $\{1, |C_4^*|\}$-CSP($\mathcal{C}_4^*$) and $\{1, |C_6|/2\}$-CSP($\mathcal{C}_6$) **do** have the same complexities (recall the reductions to Ret($\mathcal{C}_4^*$) and Ret($\mathcal{C}_6$) involved CSP($\mathcal{K}_{|C_4^*|}$) and CSP($\mathcal{K}_{|C_6|/2}$), respectively.

7 Conclusion

We have taken first important steps to understanding the complexity of CSPs with counting quantifiers, even though several interesting questions have resisted solution. We would like to close the paper with some open problems.

In Section 4.1, the case $n = 2j$ remains. When $j = 1$ and $n = 2$, we have $\{1\}$-CSP($\mathcal{K}_2$)=CSP($\mathcal{K}_2$) which is in L by [24]. For higher j, the question of the complexity of $\{j\}$-CSP($\mathcal{K}_{2j}$) is both challenging and open.

We would like to prove the following more natural variants of Theorem 4, whose involved combinatorics appear to be much harder.

Conjecture 1. Let $\mathcal{H}$ be a graph. Then

- $[2^*1^*]$-CSP($\mathcal{H}$) is in P, if $\mathcal{H}$ is a forest or a bipartite graph containing $\mathcal{C}_4$,
- $[2^*1^*]$-CSP($\mathcal{H}$) is NP-hard, if otherwise.

Conjecture 2. Let $\mathcal{H}$ be a graph. Then

- $\{1,2\}$-CSP($\mathcal{H}$) is in P, if $\mathcal{H}$ is a forest or a bipartite graph containing $\mathcal{C}_4$,
- $\{1,2\}$-CSP($\mathcal{H}$) is NP-hard, if otherwise.

References

1. Barto, L., Kozik, M., Niven, T.: The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of Bang-Jensen and Hell). SIAM Journal on Computing 38(5), 1782–1802 (2009)
2. Bodirsky, M., Kára, J., Martin, B.: The complexity of surjective homomorphism problems – a survey. CoRR abs/1104.5257 (2011)
3. 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)
4. Börner, F., Krokhin, A., Bulatov, A., Jeavons, P.: Quantified constraints and surjective polymorphisms. Tech. Rep. PRG-RR-02-11, Oxford University (2002)
5. Bulatov, A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. J. ACM 53(1), 66–120 (2006)
6. Bulatov, A., Krokhin, A., Jeavons, P.G.: Classifying the complexity of constraints using finite algebras. SIAM Journal on Computing 34, 720–742 (2005)
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.: Existentially restricted quantified constraint satisfaction. Inf. Comput. 207(3), 369–388 (2009)
9. Chen, H.: Quantified Constraint Satisfaction and the Polynomially Generated Powers Property. Algebra Universalis 65, 213–241 (2011)
10. Clark, R., Grossman, M.: Number sense and quantifier interpretation. Topoi 26(1), 51–62 (2007)
11. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory, 2nd edn. Springer (1999)
12. Feder, T., Hell, P.: List homomorphisms to reflexive graphs. J. Comb. Theory, Ser. B 72(2), 236–250 (1998)
13. Feder, T., Hell, P., Huang, J.: List homomorphisms and circular arc graphs. Combinatorica 19(4), 487–505 (1999)

14. Feder, T., Hell, P., Jonsson, P., Krokhin, A.A., Nordh, G.: Retractions to pseudoforests. SIAM J. Discrete Math. 24(1), 101–112 (2010)
15. Feder, T., Vardi, M.: The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. SIAM Journal on Computing 28, 57–104 (1999)
16. Hell, P., Nešetřil, J.: On the complexity of H-coloring. Journal of Combinatorial Theory, Series B 48, 92–110 (1990)
17. Hemaspaandra, E.: Dichotomy theorems for alternation-bounded quantified boolean formulas. CoRR cs.CC/0406006 (2004)
18. Kolaitis, P.G., Vardi, M.Y.: A logical Approach to Constraint Satisfaction. In: Finite Model Theory and Its Applications (Texts in Theoretical Computer Science. An EATCS Series). Springer-Verlag New York, Inc. (2005)
19. Lange, K.-J.: Some results on majority quantifiers over words. In: Proceedings of 19th IEEE Annual Conference on Computational Complexity, pp. 123–129 (June 2004)
20. Martin, B.: First-Order Model Checking Problems Parameterized by the Model. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.) CiE 2008. LNCS, vol. 5028, pp. 417–427. Springer, Heidelberg (2008)
21. Martin, B., Madelaine, F.: Towards a Trichotomy for Quantified *H*-Coloring. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 342–352. Springer, Heidelberg (2006)
22. Otto, M.: Bounded variable logics and counting – A study in finite models, vol. 9, IX+183 pages. Springer (1997)
23. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
24. Reingold, O.: Undirected connectivity in log-space. J. ACM 55(4), 1–24 (2008)
25. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of STOC 1978, pp. 216–226 (1978)

Space-Bounded Kolmogorov Extractors*

Daniil Musatov

Moscow Institute for Physics and Technology and
Branch for Theoretical and Applied Research, Yandex LLC
musatych@gmail.com

Abstract. An extractor is a function that receives some randomness and either "improves" it or produces "new" randomness. There are statistical and algorithmical specifications of this notion. We study an algorithmical one called Kolmogorov extractors and modify it to resource-bounded version of Kolmogorov complexity. Following Zimand we prove the existence of such objects with certain parameters. The utilized technique is "naive" derandomization: we replace random constructions employed by Zimand by pseudo-random ones obtained by Nisan-Wigderson generator.

1 Introduction

An extractor is a deterministic procedure that extracts randomness from weak random sources. Concerning finite strings there are two concepts of specifying this notion: statistical and algorithmical. The statistical one considers probability distributions on inputs and outputs of such procedure in terms of min-entropy. Loosely speaking, to extract randomness means to produce a distribution with higher min-entropy than any input has. This notion was invented by Nisan and Zuckerman in early 90s and was deeply examined by many researchers through the last two decades. An introduction to the field is presented by Shaltiel in [7].

An algorithmic counterpart, i.e. the notion of Kolmogorov extractors, was invented in the last several years (see [2], [3] and [12]). Roughly speaking, a Kolmogorov extractor is a function that receives two strings with sufficiently large Kolmogorov complexity and sufficiently small dependency and outputs a sufficiently long string having complexity closer to its length than any input has. It was shown in [3] that there exists a deep connection between ordinary and Kolmogorov extractors. Namely, each ordinary extractor is a Kolmogorov extractor with a bit worse parameters and vice versa. As shown in [10] and [11], there also exist *strong* Kolmogorov extractors in a sense that the output is rather complex even being conditioned on any single input.

The notion of Kolmogorov extractors may be naturally expanded to space-bounded complexity, it was done already in the original paper [2]. The existence

* Supported by ANR Sycomore, NAFIT ANR-08-EMER-008-01 and RFBR 09-01-00709-a grants.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 266–277, 2012.

results of that paper hold both for common and space-bounded Kolmogorov extractors. For the unbounded case these resuts were improved by Zimand in [10] and [11]. In this paper we convert Zimand's results to the space-bounded case and hence improve the respective results of Fortnow et al. Since Zimand's construction is not efficient, this conversion cannot be done straightforwardly. The technique we employ is the "naive derandomization" method introduced in [6] and [4] and later used in [13] and [14]. Originally, Zimand have characterized Kolmogorov extractors by some combinatorial properties. The existence of an object with such properties was proven implicitly. We show that such an object may be found in the output of Nisan-Wigderson pseudo-random generator. That is, to find a required object one does not need to search through all possible objects but needs only to check all seeds of the generator. This crucially decreases the required space from exponential to polynomial.

The rest of the paper is organized as follows. In Sect. 2 we give formal definitions of all involved objects and formulate necessary results. In Sect. 3 we give formal definitions for space-bounded Kolmogorov extractors, formulate our existence theorems, outline the proof idea and present detailed proofs.

2 Preliminaries

2.1 Kolmogorov Complexity

Let $\mathcal{V}$ be a two-argument Turing machine. We refer to the first argument as to the "program" and to the second argument as to the "argument". (Plain) Kolmogorov complexity of a string x conditioned on y with respect to $\mathcal{V}$ is the length of a minimal $\mathcal{V}$-program p that transforms y to x, i.e.

$$\mathrm{C}_{\mathcal{V}}(x \mid y) = \min\{p\colon \mathcal{V}(p, y) = x\}$$

There exists an optimal machine $\mathcal{U}$ that gives the smallest complexity up to an additive term. Specifically, $\forall \mathcal{V} \exists c \forall x, y\, \mathrm{C}_{\mathcal{U}}(x|y) < \mathrm{C}_{\mathcal{V}}(x|y) + c$. We employ such a machine $\mathcal{U}$, drop the subscript and formulate all theorems up to a constant additive term. The unconditional complexity $\mathrm{C}(x)$ is the complexity with empty condition $\mathrm{C}(x \mid \varepsilon)$, or the length of a shortest program *producing* x.

The next notion to be defined is resource-bounded Kolmogorov complexity. Loosely speaking, it is the length of a minimal program that transforms y to x efficiently. Formally, Kolmogorov complexity of a string x conditioned on y in time t and space s with respect to $\mathcal{V}$ is the length of a shortest program p such that $\mathcal{V}(p, y)$ outputs x, works in t steps and uses s cells of memory. This complexity is denoted by $\mathrm{C}^{t,s}_{\mathcal{V}}(x \mid y)$. Here the choice of $\mathcal{V}$ alters not only complexity, but also time and space bounds. Specifically, the following theorem holds:

Theorem 1. *There exist a machine $\mathcal{U}$ such that for any machine $\mathcal{V}$ there exists a constant c such that for all x, y, s and t it is true that $\mathrm{C}^{s,t}_{\mathcal{U}}(x \mid y) \leq \mathrm{C}^{cs,ct\log t}_{\mathcal{V}}(x \mid y) + c$.*

In our paper we deal only with space bounds, so we drop the time-bound superscript in all notations.

2.2 Extractors

A k-weak random source of length n is a random variable distributed on $\{0,1\}^n$ that has min-entropy not less than k, that is, any particular string occurs with probability not greater than 2^{-k}. The statistical distance between two randomness distributions ξ and η on the same set T is $\max_{S \subset T} |\xi(S) - \eta(S)|$.

Loosely speaking, a randomness extractor is a procedure that converts weak random sources to nearly uniform random sources. There are two common specifications of this notion: seeded extractor that gets a weak random source and a (small) truly random source and multi-source extractor that gets two weak random sources. The latter one is relevant to our paper, so we define it formally. A multi-source extractor with parameters (n, m, k, ε) is a function $\mathrm{Ext}\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ such that for any two independent k-weak random sources x and y the induced distribution $\mathrm{Ext}(x,y)$ is ε-close to uniform. A multi-source extractor may be considered as a $2^n \times 2^n$ table with each cell coloured in one of 2^m colours. It may be proven that the extractor property is equivalent to the following: for any set of colours ("palette") $A \subset \{0,1\}^m$ in any rectangle $S_1 \times S_2$, where $S_i \subset \{0,1\}^n$ and $|S_i| \geq 2^k$ the fraction of cells coloured in a colour from A differs from $|A|/2^m$ by at most ε. Yet another equivalent definition is the following: for any $Q \in [1, 2^m]$ and any $S_1 \times S_2$ the fraction of cells coloured in Q most popular colours does not exceed $Q/2^m + \varepsilon$.

2.3 Balanced Tables

A balanced table is a combinatorial object considered by Zimand in papers [9], [10] and [11]. In the last paper he has also introduced a slightly different object called rainbow balanced table. In some sense they are similar to multi-source extractors but use another notion of closeness of distributions. Here we present alternative definitions that seem more comprehensive though equivalent to original ones.

A (K,Q)-balanced table is a function $\mathrm{BT}\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ with the following property: in any rectangle $S_1 \times S_2$, where $S_i \subset \{0,1\}^n$ and $|S_i| \geq K$ the fraction of cells coloured in Q most popular colours is less than $2Q/2^m$. Contrasting to multi-source extractors, this property is less restrictive for big palettes ($|A| > \varepsilon 2^m$) and more restrictive for small ones ($|A| < \varepsilon 2^m$).

In [11] Zimand introduces a variation of the above-defined object named rainbow balanced table. Here we present a bit different though equivalent definition of it. A (K,Q)-rainbow balanced table is a function $\mathrm{RBT}\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ with the following property. Consider a rectangle $S_1 \times S_2$ where $S_i \subset \{0,1\}^n$ and $|S_i| \geq K$. Let us mark in each row all cells coloured in one of Q most popular colours of this row. Call the table row-rainbow-balanced if the fraction of marked cells in any rectangle is less than $2Q/2^m$. Then do the same with columns and call the table column-rainbow-balaced if the fraction of marked cells is again less than $2Q/2^m$. Finally, call the table rainbow-balanced if it is both row- and column-rainbow-balaced.

It was shown by the probabilistic argument that there exist balanced tables with $K \geq \sqrt{M}\,\mathrm{poly}(n)$ and $Q \geq \sqrt{M}\,\mathrm{poly}(n)$ and rainbow balanced tables with $K \geq M\,\mathrm{poly}(n)$ and any Q.

2.4 Nisan-Wigderson Generators

The Nisan-Wigderson pseudo-random generator is a deterministic polynomial-time function that generates n pseudo-random bits from polylog(n) truly random bits. The output of such generator cannot be distinguished from truly random string by small circuits. Specifically, we exploit the following theorem from [5]:

Theorem 2. *For any constant d there exists a family of functions $G_n\colon \{0,1\}^k \to \{0,1\}^n$, where $k = O(\log^{2d+6} n)$, such that two properties hold:*

Computability: *G is computable in workspace* $\mathrm{poly}(k)$ *(that is, any particular bit of $G(x)$ may be found in this space);*

Indistinguishability: *For any family of circuits C_n of size* $\mathrm{poly}(n)$ *and depth d for any positive polynomial p for all large enough n it holds that:*

$$|\mathrm{Prob}_x\{C_n(G_n(x)) = 1\} - \mathrm{Prob}_y\{C_n(y) = 1\}| < \frac{1}{p(n)},$$

where x is distributed uniformly in $\{0,1\}^k$ and y — in $\{0,1\}^n$.

By rescaling parameters we get the following

Corollary 1. *For any constant d there exists a family of functions $G_n\colon \{0,1\}^k \to \{0,1\}^N$, where $k = \mathrm{poly}(n)$ and $N = 2^{\mathrm{poly}(n)}$, such that two properties hold:*

- *G is computable in polynomial workspace;*
- *For any family of circuits C_n of size $2^{\mathrm{poly}(n)}$ and depth d, for any constant c and for all large enough n it holds that:*

$$|\mathrm{Prob}_x\{C_n(G_n(x)) = 1\} - \mathrm{Prob}_y\{C_n(y) = 1\}| < 2^{-cn}.$$

The last corollary implies the following basic principle:

Lemma 1. *Let $\mathcal{C}_n$ be some set of combinatorial objects encoded by boolean strings of length $2^{O(n)}$. Let $\mathcal{P}$ be some property satisfied for fraction at least α of objects in $\mathcal{C}_n$ that can be tested by a family of circuits of size $2^{O(n)}$ and constant depth. Then for sufficiently large n the property $\mathcal{P}$ is satisfied for fraction at least $\alpha/2$ of values of G_n, where G_n is the function from the previous corollary.*

2.5 Constant-depth Circuits for Approximate Counting

It is well-known that constant-depth circuits cannot compute the majority function. All the more they cannot compute a general threshold function that equals 1 if and only if the fraction of 1's in its input exceeds some threshold α. Nevertheless, one can build such circuits that compute threshold functions approximately. Namely, the following theorem holds:

Theorem 3 ([1], [8]). *Let $\alpha \in (0,1)$. Then for any (constant) ε there exists a constant-depth and polynomial-size circuit C such that $C(x) = 0$ if the fraction of 1's in x is less than $\alpha - \varepsilon$ and $C(x) = 1$ if the fraction of 1's in x is greater than $\alpha + \varepsilon$.*

Note that nothing is promised if the fraction of 1's is between $\alpha - \varepsilon$ and $\alpha + \varepsilon$. So, the fact that $C(s) = 0$ guarantees only that the fraction of 1's is at most $\alpha + \varepsilon$, and $C(s) = 1$ — that it is at least $\alpha - \varepsilon$.

3 Main Result

3.1 Overview

In this section we give all necessary definitions, observe existing results and formulate our theorem.

Let us formally define a Kolmogorov extractor. Dependency between x and y is defined as $\mathrm{dep}(x, y) = \mathrm{C}(x) + \mathrm{C}(y) - \mathrm{C}(x, y)$. A computable function $\mathrm{KExt}\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ is a (k, δ)-Kolmogorov extractor if for any x and y of length n if $\mathrm{C}(x) > k$, $\mathrm{C}(y) > k$ and $\mathrm{dep}(x, y) < \delta$ then $\mathrm{C}(\mathrm{KExt}(x, y)) > m - \delta - O(\log n)$. Say that KExt is a strong (k, δ)-Kolmogorov extractor if, moreover, $\mathrm{C}(\mathrm{KExt}(x, y)|x) > m - \delta - O(\log n)$ and $\mathrm{C}(\mathrm{KExt}(x, y)|y) > m - \delta - O(\log n)$.

Zimand has proven that there exist Kolmogorov extractors with parameters close to optimal:

Theorem 4 ([10], [11]). *Let $k(n)$ and $\delta(n)$ be computable functions, such that $1 < k(n) < n$ and $1 < \delta(n) < k(n) - O(\log n)$. Then there exists a $(k(n), \delta(n))$-Kolmogorov extractor for $m = 2k(n) - O(\log n)$ and a strong $(k(n), \delta(n))$-Kolmogorov extractor for $m = k(n) - O(\log n)$.*

We rewrite the definitions and the theorem in the case of space-bounded complexity. Since the difference $\mathrm{C}^s(x) - \mathrm{C}^s(x|y)$ is not monotone in s, we get rid of explicit usage of the term "dependency". Instead we say that a computable function KExt is a (k, δ)-Kolmogorov extractor with a space bound $s = s(n)$ if KExt is computable in space $O(s(n))$ and for some constant $\mu > 1$ (not dependent on x, y, k, s, but possibly dependent on KExt) if $\mathrm{C}^s(x) > k$, $\mathrm{C}^s(y) > k$, and $\mathrm{C}^{\mu s}(x, y) > \mathrm{C}^s(x) + \mathrm{C}^s(y) - \delta$ then $\mathrm{C}^s(\mathrm{KExt}(x, y)) > m - \delta - O(\log n)$. If, moreover, $\mathrm{C}^s(\mathrm{KExt}(x, y)|x) > m - \delta - O(\log n)$ and $\mathrm{C}^s(\mathrm{KExt}(x, y)|y) > m - \delta - O(\log n)$ then the Kolmogorov extractor is strong.[1] We increase the space limit from s to μs in the definition of "dependency" since the space limit is determined up to a multiplicative constant dependent on the description method. We prove the following:

Theorem 5. *There exists a polynomial $p(n)$ such that for any space-constructible function $s(n) > p(n)$ and any computable in space $s(n)$ functions $1 < k(n) < n$ and $1 < \delta(n) < k(n) - O(\log n)$ there exists a $(k(n), \delta(n))$-Kolmogorov extractor with space bound $s(n)$ for $m = 2k(n) - O(\log n)$ and a strong $(k(n), \delta(n))$-Kolmogorov extractor for $m = k(n) - O(\log n)$.*

[1] Fortnow et al. do use the term "dependency" in [2] but define it for two distinct space bounds that correspond to s and μs in our definition.

3.2 Proof Idea

In this section we retell Zimand's argument in space-bounded environment and emphasize what must be added to complete the proof.

We show that a certain balanced table is in fact a Kolmogorov extractor. The main idea is to obtain a contradiction between the hardness of (x, y) and the simplicity of $T(x, y)$ by employing the balancing property. Let us come to more details. Let $d = \delta + c \log n$, where c is a constant to be determined later. Take a $(2^k, 2^{m-d})$-balanced table $T\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$. Let $B_x = \{z \mid \mathrm{C}^s(z) \leq \mathrm{C}^s(x)\}$, $B_y = \{z \mid \mathrm{C}^s(z) \leq \mathrm{C}^s(y)\}$ and $A = \{w \mid \mathrm{C}^s(w) < m - d\}$. Obviously, $(x, y) \in B_x \times B_y$. By the balancing property, the set $B_x \times B_y$ contains less than $2 \cdot 2^{\mathrm{C}^s(x)+1} 2^{\mathrm{C}^s(y)+1}/2^d$ cells coloured in a colour from A, that is, in a colour with complexity less than $m - d$. (If necessary, expand B_x, B_y and A arbitrarily to obtain sets of required size and apply the balancing property). Hence, (x, y) may be described by the table T, the sets B_x, B_y and A, and the ordinal number of (x, y) among cells in $B_x \times B_y$ coloured in a colour from A. By the balancing property the last number requires at most $\mathrm{C}^s(x) + \mathrm{C}^s(y) - d + 3$ bits. The sets B_x, B_y and A are described completely by n, $\mathrm{C}^s(x)$ and $\mathrm{C}^s(y)$ and may be enumerated in space $O(s)$. If we obtain a table T that has complexity $O(\log n)$ and may be evaluated in space $O(s)$ then we have $\mathrm{C}^{O(s)}(x, y) < \mathrm{C}^s(x) + \mathrm{C}^s(y) - d + O(\log n)$ that contradicts the assumption $\mathrm{C}^{\mu s}(x, y) > \mathrm{C}^s(x) + \mathrm{C}^s(y) - \delta$ for μ and c large enough. So, the crucial missing component is a simple and space-efficiently computable balanced table. Without a space bound the existence of such table is proven by the probabilistic method and a simple table may be found by an exhaustive search: the first table in some canonical order has small complexity. Having a space bound added the construction should be derandomized.

To get the strong Kolmogorov extractor property we need to replace a balanced table by a rainbow balanced one with parameters $(2^k, Q = 2^{m-d})$ where again $d = \delta + c \log n$. Let A_v be the set $\{w \mid \mathrm{C}^s(w|v) < m - d\}$. Obviously, $|A_v| < Q$. Call a string v *bad* if the fraction of cells in the row $B_x \times \{v\}$ coloured in a colour from A_v is greater than 2×2^{-d}. The fraction of cells coloured in one of Q most popular colours is even greater, so there are less than K bad rows, since otherwise the colouring of the rectangle $B_x \times \{\text{bad rows}\}$ contradicts the rainbow balancing property. If one could enumerate bad rows in space s then all bad rows would have complexity at most k and so y would be a good row. If $T(x, y) \in A_y$ and y is known then x may be described by the table T, sets B_x and A_y and the ordinal number of x among all cells in the row y coloured in a colour from A_y. Since y is good the last number requires less than $C^s(x) - d + 1$ bits. The sets B_x and A_y are described completely by n and $\mathrm{C}^s(x)$ and may be enumerated in space $O(s)$. Finally, if T has complexity $O(\log n)$ and may be evaluated in space $O(s)$ then we obtain a contradiction similar to the previous one. The whole argument may be repeated symmetrically for the complexity conditioned on x. As before, the crucial missing component is a simple and space-efficiently computable rainbow balanced table that also allows to enumerate space-efficiently "bad" rows and columns. We now turn to a high-level description of our derandomization method.

3.3 Derandomization Plan

To derandomize the construction we use a "naive" idea of replacing a random construction by a pseudo-random one. This idea was originally presented in [4] and [6]. The essence of the idea is to replace a brute-force search among all possible objects by a brute-force search in the output of the Nisan-Wigderson pseudo-random generator. Since the length of the seed is polylogarithmic in the size of the output the range of the search decreases crucially. To make the things work we should, firstly, prove that the necessary object exists among the output of the NW-generator and, secondly, prove that a good seed for the generator may be found efficiently. To prove the first thing we employ the basic principle 1 that involves a constant-depth circuit to test the balancing property. The original balancing properties seem to be too hard to be tested by such circuits, so we weaken them. Specifically, for ordinary balanced tables we limit the balancing condition only to those rectangles and palettes being actually used in the proof. For rainbow balanced tables we go even further and directly specify the property used in the proof. Details follow in the next several subsections. In Sect. 3.4 we specify the weakening of the balancing property. In Sect. 3.5 we prove that a modified balanced table exists in the output of the NW-generator. Next, in Sect. 3.6 we show how to find a good seed in limited space. Finally, in Sect. 3.7 we put all things together and finish the proof. We do all steps simultaneously for balanced tables (leading to Kolmogorov extractors) and rainbow balanced tables (leading to strong Kolmogorov extractors).

3.4 The Weakening of Balancing Conditions

Recall that a table is a function $T\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$. We refer to the first argument as to 'column', to the second one as to 'row' and to the value as to 'colour'. Let b be a positive number and $k < n$ and $q < m$ be some integers. Let there be a system $\mathcal{S}$ of pairs (S, l) where S is a subset of $\{0,1\}^n$ and $l \in [k, n]$ such that for any pair the set S contains less than 2^l elements and the whole system contains $2^{\mathrm{poly}(n)}$ pairs. Let there also be a system $\mathcal{Q}$ of subsets of $\{0,1\}^m$ (i.e., palettes) such that any $Q \in \mathcal{Q}$ contains less than 2^q elements and the whole system contains $2^{\mathrm{poly}(n)}$ sets. Later we refer to sets in such systems as to *relevant* ones. Say that a table T is $(b, \mathcal{Q}, \mathcal{S})$-balanced if for any (S_1, l_1) and $(S_2, l_2) \in \mathcal{S}$ and for any $Q \in \mathcal{Q}$ the number of cells in $S_1 \times S_2$ coloured in a colour from Q is less than $2^{l_1+l_2+q-m+b}$. If the sizes of S_1 and S_2 are maximal, Q is the set of 2^q most popular colours and $b = 1$ then the bound matches the original one, i.e. the fraction of the popular colours is at most $2 \times \frac{2^q}{2^m}$. The new parameter b is introduced due to technical reasons and will be used in the next subsection. Take in mind that $b \approx 1$.

The definition of rainbow balanced tables is modified in a more complicated way: we again fix a number b and a system $\mathcal{S}$. Instead of a system $\mathcal{Q}$ of palettes we fix a system $\mathcal{R}$ of tuples of palettes. Here each palette contains less than 2^q elements, each tuple has length 2^l for some $l \in [k, n]$ and the whole system contains $2^{\mathrm{poly}(n)}$ tuples. Take arbitrary sets S_1 and $S_2 \in \mathcal{S}$ with corresponding

l_1 and l_2 and a tuple $\boldsymbol{Q} = (Q_1, \ldots, Q_{2^{l_2}}) \in \mathcal{R}$. Then for each $i \in [1, |S_2|]$ mark those cells in i-th row of $S_1 \times S_2$ coloured in a colour from Q_i. Say that a row is *saturated* if it contains more than $2^{l_1+q-m+b}$ marked cells. We say that a table T is $(b, \mathcal{R}, \mathcal{S})$-row rainbow balanced if for any S_1, S_2 and $\boldsymbol{Q}$ the total number of marked cells in saturated rows is less than $2^{l_1+q-m+k+b}$. (In particular, there are less than 2^k saturated rows). We define a $(b, \mathcal{R}, \mathcal{S})$-column rainbow balanced table similarly and say that a table is $(b, \mathcal{R}, \mathcal{S})$-rainbow balanced if it is both $(b, \mathcal{R}, \mathcal{S})$-row and $(b, \mathcal{R}, \mathcal{S})$-column rainbow balanced.

One may easily see that for $b \geq 1$ our modifications actually do strictly weaken both balancing properties, so a random table satisfies a modified property with even greater probability than the original one.

3.5 Existence of Balanced Tables in the Output of the NW-generator

To prove that a modified (rainbow) balanced table exists in the output of the NW-generator we employ the basic principle 1. We present a constant-depth exponential circuit that tests the modified balancing property. By the basic principle, since the generator fools such circuits and a random table satisfies the modified balancing property with positive probability, the same holds for a pseudo-random one. A construction of such a circuit follows.

The modified balancing properties are tested rather straightforwardly. Since we do not need to build a uniform circuit we just hardwire the lists of relevant sets into the circuit. That is, we construct a circuit for a particular tuple (S_1, S_2, l_1, l_2, Q) (or $(S_1, S_2, l_1, l_2, \boldsymbol{Q})$), make $2^{\mathrm{poly}(n)}$ copies of this circuit for different tuples and take a conjunction. The construction of such a circuit follows.

It is rather easy to check whether a particular cell in $S_1 \times S_2$ is marked. Indeed, we should check whether its colour coincides with one of those belonging to Q (or Q_i) and take the disjunction of all results. The difficult point is to count the number of marked cells and to compare this number to $2^{l_1+l_2+q-m+b}$. (For rainbow balance we need to count marked cells in saturated rows and columns). This task cannot be solved exactly by constant-depth circuits but may be solved approximately. Fortunately, approximate solution is enough for our goal.

We employ a circuit existing by theorem 3. Specifically, for ordinary balanced tables this curcuit has $|S_1| \cdot |S_2|$ inputs, outputs 1 if there are less than $2^{l_1+l_2+q-m+1}$ ones among the inputs, outputs 0 if there are more than $2^{l_1+l_2+q-m+1.01}$ ones among the inputs and outputs any value otherwise. For rainbow balanced tables we use two such circuits sequentially. Firstly, we apply to every row a circuit with $|S_1|$ inputs that outputs 1 if there are less than $2^{l_1+q-m+1}$ ones among its inputs, outputs zero if there are more than $2^{l_1+q-m+1.01}$ ones and outputs any value otherwise. Secondly, we count the number of ones in rows that produce one on the previous stage. To this end, we take conjunctions of the output of the previous circuit with the inpits and apply a circuit with $|S_1| \cdot |S_2|$ inputs that returns 1 if it receives less than $2^{l_1+q-m+k+1}$ ones, returns 0 if it receives more than $2^{l_1+q-m+k+1.01}$ ones and returns any value otherwise. This construction is repeated for columns and a conjunction of two values is taken.

The last subcircuit completes the description. Let us sketch the structure of the whole circuit once more. The input specifies colours of all 2^{2n} cells of the table. We also add constants for all possible colours from $\{0,1\}^m$. The full circuit consists of $2^{\mathrm{poly}(n)}$ identical blocks. Each block has two groups of inputs. The left group specifies colours of all cells in a particular rectangle $S_1 \times S_2$. The right group specifies a palette Q (or a set of palettes $\boldsymbol{Q}$). Different blocks are hardwired to different inputs and constants. For ordinary balanced tables each block consists of two levels. On the first level a simple equivalence circuit is applied to every pair of a colour from the left and a colour from the right. On the second level an approximate counting circuit is applied to the outputs of the first level. For rainbow balanced tables each block consists of five levels. The first level is the same. On the second level an approximate counting circuit is applied to the outputs of the first level separately for each row and for each column. On the third a conjunction of the outputs of the first two levels is taken separately for each row and for each column. On the fourth level another approximate counting circuit is applied to the outputs of the third level, separately for rows and columns. On the fifth level a conjunction of two results of the fourth level is taken. Finally, a conjunction is applied to outputs of all blocks.

Clearly, these circuits have exponential size and constant depth. It is also clear that these circuits return one on (rainbow) balanced tables. Hence, they return one with positive probability on a random table. Hence, they return one with positive probability on a pseudo-random table produced by the NW-generator. If the first (resp., second) circuit returns one then the table is balanced (resp., rainbow balanced) for parameter $b = 1.01$. In the next two subsections we specify the systems $\mathcal{S}$, $\mathcal{Q}$ and $\mathcal{R}$, show how to find a good seed for the generator and how to use a 1.01-balanced and 1.01-rainbow balanced tables to obtain the result.

3.6 Searching for a Good Seed

Until this point the construction was valid for any choice of the systems $\mathcal{S}$, $\mathcal{Q}$ and $\mathcal{R}$. The searching for a good seed cannot be performed for arbitrary systems, so now we specify them.

We take $\mathcal{S}$ to be the system of all pairs $(\{z \mid \mathrm{C}^s(z) < l\}, l)$ for $s = s(n)$ and $l \in [k, n]$. We take $\mathcal{Q}$ to be the system of all sets $\{z \mid \mathrm{C}^s(z) < q\}$ for $s = s(n)$ and $q \in [1, m]$. Finally, we take $\mathcal{R}$ to be the system of all tuples $(\{z \mid \mathrm{C}^s(z|v_1) < q\}, \ldots, \{z \mid \mathrm{C}^s(z|v_{2^l}) < q\})$ where $(\{v_1, \ldots, v_{2^l}\}, l) \in \mathcal{S}$. Clearly, the sizes of all sets, tuples and systems satisfy the requirements. Since $s(n)$ is space-constructible, the systems $\mathcal{S}$ and $\mathcal{Q}$ are enumerable in space $O(s)$. It means that there exists an $O(s)$-space algorithm that gets two numbers i and j and returns the i-th element of the j-th set in $\mathcal{S}$ (or $\mathcal{Q}$). If one of the numbers is out of range the algorithm returns an error message. A similar statement holds for $\mathcal{R}$, here the enumerating algorithm gets three numbers: the number of the tuple, the number of the set in the tuple and the number of the element in the set.

Since we care only about space, the problem of searching a good seed is equivalent to the problem of checking whether a seed is good. The crucial property of NW-generator that makes such a check possible in small space is that any bit of

the output may be computed in polynomial space independently from all other bits. That is, one need not store the whole exponential output to check some local property. A detailed description of such a check follows.

After this point the constructions for ordinary and rainbow balanced tables are rather different. We start with the construction for ordinary tables. Firstly, let us notice that a seed is good if and only if it is good for any tuple (S_1, S_2, l_1, l_2, Q). So, it is sufficient to sequentially check that a seed is good for any such tuple. A tuple is determined by the ordinal numbers of (S_1, l_1) and (S_2, l_2) in the enumeration of $\mathcal{S}$ and that of Q in the enumeration of $\mathcal{Q}$. Having these numbers fixed, we sequentially generate colours of all cells in $S_1 \times S_2$ and compare them to all colours in Q. Count the number of successive comparisons. Say that the tuple is good if this number is less than $2^{l_1+l_2+q-m+1.01}$. Since we are no more restricted to constant-depth circuits and may instead use any space-bounded computations the counting is made precisely. Since $\mathcal{S}$ and $\mathcal{Q}$ are enumerable in space $O(s)$, the generator uses another $O(s)$ portion of space, only space $O(n)$ is used for intermediate storage and s is at least polynomial in n, the total space requirement sums up to $O(s)$.

For rainbow balanced tables we sequentially check that a seed is good for any tuple (S_1, S_2, l_1, l_2). The sets of palettes for row and column rainbow balanced properties are generated from S_1 and S_2 respectively: for each $S_1 = \{x_1, \ldots, x_L\}$ and $S_2 = \{y_1, \ldots, y_M\}$ we take $\boldsymbol{Q}_1 = (\{z \mid \mathrm{C}^s(z|y_1) < q\}, \ldots, \{z \mid \mathrm{C}^s(z|y_M) < q\})$ and $\boldsymbol{Q}_2 = (\{z \mid \mathrm{C}^s(z|x_1) < q\}, \ldots, \{z \mid \mathrm{C}^s(z|x_L) < q\})$. The subsequent check is performed by direct counting, as in the previous algorithm. The difference is that the counting proceeds in two stages: for any row (or column) the number of marked cells is counted, then it is determined whether the row (column) is saturated and the numbers for saturated rows (columns) are summed up. The used space is again $O(s)$.

3.7 Completion of the Proof

In this section we prove that the tables generated from seeds found by two algorithms in the previous subsection are indeed Kolmogorov and strong Kolmogorov extractors respectively.

Firstly we check the Kolmogorov extractor property. Fix the extractor parameters $k = k(n)$, $\delta = \delta(n)$ and $s = s(n)$. Let $d = \delta + c \log n$ where c is a constant to be determined later and let $q = m - d$. Let p be the seed found for parameters k and q in Sect. 3.6. We want to prove that $\mathrm{KExt}_p = NW(p)$ is a (k, δ)-Kolmogorov extractor for space bound s. Firstly, note that it is computable in space $O(s)$: this space is enough both for finding p and computing $\mathrm{KExt}_p(x, y)$. Secondly, let us prove the Kolmogorov extractor property. Take two strings x and y such that $\mathrm{C}^s(x) > k$, $\mathrm{C}^s(y) > k$ and $\mathrm{C}^{\mu s}(x, y) > \mathrm{C}^s(x) + \mathrm{C}^s(y) - \delta$ where μ does not depend on x or y and will be determined later. To obtain a contradiction assume that $\mathrm{C}^s(\mathrm{KExt}_p(x, y)) < m - d$. Denote $l_1 = \mathrm{C}^s(x)$ and $l_2 = \mathrm{C}^s(y)$ and consider the sets $S_1 = \{z \mid \mathrm{C}^s(z) \leq l_1\}$ and $S_2 = \{z \mid \mathrm{C}^s(z) \leq l_2\}$. Each of them is relevant by construction. Denote $Q = \{z \mid \mathrm{C}^s(z) < q\}$. This set is also relevant. By the choice of p the rectangle $S_1 \times S_2$ contains less than $2^{l_1+l_2+q-m+1.01}$

cells coloured in one of colours from Q. By the assumption and the definition of S_1 and S_2, the pair (x, y) belongs to these cells. In this case (x, y) may be described by n, l_1, l_2, q and the ordinal number of (x, y) among these cells. Indeed, having n known we may find a good seed p; having l_1, l_2 and q known we may search through cells $S_1 \times S_2$ and check whether the current one has a colour from Q. The ordinal number specifies the needed cell. The total required space is $O(s)$. The total number of used bits is $l_1 + l_2 - d + O(\log n)$. So, we obtain $\mathrm{C}^{O(s)}(x, y) < \mathrm{C}^s(x) + \mathrm{C}^s(y) - d + O(\log n) = \mathrm{C}^s(x) + \mathrm{C}^s(y) - \delta - c \log n + O(\log n)$ that contradicts the condition $\mathrm{C}^{\mu s} > \mathrm{C}^s(x) + \mathrm{C}^s(y) - \delta$ for μ and c taken large enough.

Next, we check the strong Kolmogorov extractor property. We again fix the extractor parameters k, δ and s. As before, let $d = \delta + c \log n$ and $q = m - d$. Let p be again the seed found for parameters k and q (and the rainbow balancing property) in Sect. 3.6. We want to prove that $\mathrm{KExt}_p = NW(p)$ is a strong (k, δ)-Kolmogorov extractor for space bound s. The computability in space $O(s)$ is again easily obtained. Prove the strong Kolmogorov extractor property. Take two strings x and y such that $\mathrm{C}^s(x) = l_1 > k$, $\mathrm{C}^s(y) = l_2 > k$ and $\mathrm{C}^{\mu s}(x, y) > l_1 + l_2 - \delta$ where μ does not depend on x or y and will be determined later. To obtain a contradiction assume that $\mathrm{C}^s(\mathrm{KExt}_p(x, y)|y) < m - d$. Thus, the cell (x, y) is marked. Consider two cases: there are more than $2^{l_1 - d + 1.01}$ strings $z \in S_1$ such that $\mathrm{C}^s(\mathrm{KExt}_p(z, y)|y) < m - d$ and there are not more than $2^{l_1 - d + 1.01}$ such strings. In the first case the row y contains more than $2^{l_1 + q - m + 1.01}$ marked cells and thus is saturated. By the 1.01-rainbow balancing property the total number of marked cells in saturated rows is less than $2^{l_1 + q - m + k + 1.01} < 2^{l_1 + l_2 - \delta - c \log n}$. Then (x, y) may be described in space $O(s)$ by its ordinal number among marked cells in saturated rows and numbers n, l_1, l_2, q. Thus for large enough μ and c the complexity of (x, y) is less than $l_1 + l_2 - \delta$ that contradicts the assumption. In the second case the pair (x, y) may be described by a description of y (l_2 bits), the ordinal number of (x, y) among marked cells ($l_1 - d + 1.01$ bits) and numbers n, l_1, l_2, q ($O(\log n)$ bits), totaling to $l_1 + l_2 - d + O(\log n)$ bits. The required space is $O(s)$, so we obtain a similar contadiction to the assumption that $\mathrm{C}^{\mu s}(x, y) > l_1 + l_2 - \delta$ for large enough μ and c. This contradiction finishes the proof.

Acknowledgments. I want to thank my colleagues and advisors Andrei Romashchenko, Alexander Shen and Nikolay Vereshchagin for stating the problem and many useful comments. I also want to thank two anonymous referees for careful reading and precise comments. I am grateful to participants of seminars in Moscow State University for their attention and thoughtfulness.

References

1. Ajtai, M.: Approximate counting with uniform constant-depth circuits. In: Advances in Computational Complexity Theory, pp. 1–20. American Mathematical Society, Providence (1993)

2. Fortnow, L., Hitchcock, J., Pavan, A., Vinodchandran, N.V., Wang, F.: Extracting Kolmogorov complexity with applications to dimension zero-one laws. Information and Computation 209(4), 627–636 (2011); Preliminary version appeared in Proceedings of the 33rd International Colloquium on Automata, Languages, and Programming. LNCS, vol. 4051, pp. 335–345 (2006)
3. Hitchcock, J., Pavan, A., Vinodchandran, N.: Kolmogorov complexity in randomness extraction. Electronic Colloquium on Computational Complexity (ECCC) 16, 71 (2009)
4. Musatov, D.: Improving the Space-Bounded Version of Muchnik's Conditional Complexity Theorem via "Naive" Derandomization. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 64–76. Springer, Heidelberg (2011)
5. Nisan, N., Wigderson, A.: Hardness vs. Randomness. Journal of Computer and System Sciences 49(2), 149–167 (1994)
6. Romashchenko, A.: Pseudo-random Graphs and Bit Probe Schemes with One-Sided Error. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 50–63. Springer, Heidelberg (2011)
7. Shaltiel, R.: An Introduction to Randomness Extractors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 21–41. Springer, Heidelberg (2011)
8. Viola, E.: Randomness buys depth for approximate counting. In: Proceedings of IEEE FOCS 2011, pp. 230–239 (2011)
9. Zimand, M.: Two Sources Are Better Than One for Increasing the Kolmogorov Complexity of Infinite Sequences. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) CSR 2008. LNCS, vol. 5010, pp. 326–338. Springer, Heidelberg (2008)
10. Zimand, M.: Extracting the Kolmogorov complexity of strings and sequences from sources with limited independence. In: Proceedings 26th STACS, pp. 697–708 (2009)
11. Zimand, M.: Impossibility of Independence Amplification in Kolmogorov Complexity Theory. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 701–712. Springer, Heidelberg (2010)
12. Zimand, M.: Possibilities and impossibilities in Kolmogorov complexity extraction. Sigact News 41(4), 74–94 (2010)
13. Zimand, M.: Symmetry of information and bounds on nonuniform randomness extraction via Kolmogorov extractors. In: Proceedings of 26th IEEE Conference in Computational Complexity, pp. 148–156 (2011)
14. Zimand, M.: On the Optimal Compression of Sets in PSPACE. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 65–77. Springer, Heidelberg (2011)

Some Results on more Flexible Versions of Graph Motif

Romeo Rizzi[1] and Florian Sikora[2,3]

[1] DIMI, Università di Udine, Italy
romeo.rizzi@uniud.it
[2] Université Paris-Est, LIGM - UMR CNRS 8049, France
[3] Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Germany
florian.sikora@uni-jena.de

Abstract. The problems studied in this paper originate from Graph Motif, a problem introduced in 2006 in the context of biological networks. Informally speaking, it consists in deciding if a multiset of colors occurs in a connected subgraph of a vertex-colored graph. Due to the high rate of noise in the biological data, more flexible definitions of the problem have been outlined. We present in this paper two inapproximability results for two different optimization variants of Graph Motif. We also study another definition of the problem, when the connectivity constraint is replaced by modularity. While the problem stays NP-complete, it allows algorithms in FPT for biologically relevant parameterizations.

1 Introduction

A recent field in bioinformatics focuses in biological networks, which represent interactions between different elements (*e.g.* between amino acids, between molecules or between organisms) [1]. Such a network can be modeled by a vertex-colored graph, where nodes represent elements, edges represent interactions between them and colors give functional informations on the graph nodes. Using biological networks allows a better characterization of species, by determining small recurring subnetworks, often called *motifs*. Such motifs can correspond to a set of nodes realizing a same function, which may have been evolutionary preserved [22]. It is thus crucial to determine these motifs to identify common elements between species and transfer the biological knowledge.

Historically, motifs were defined by a set of nodes labels with a given topology (*e.g.* a path, a tree, a graph). The algorithmic problem was thus to find an occurrence of the motif in the network which respect both the label set and the given topology. This leads to problems roughly equivalent to subgraph isomorphism, a computationally difficult problem. However, in metabolic networks, similar topology can represent very different functions [17]. Moreover, in protein-protein interactions (PPI) networks, informations about the topology of motifs is often missing [5]. There is also a high rate of false positive and false negative in such networks [10]. Therefore, in some situations, topology is irrelevant, which leads to search for *functional* motifs instead of *topological* ones. In this

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 278–289, 2012.

setting, we still ask for the conservation of the node labels, but we replace topology conservation by the weaker requirement that the subnetwork should form a connected subgraph of the target graph. This approach was proposed by Lacroix *et al.*, defining Exact Graph Motif [17].

- **Input**: A graph $G = (V, E)$, a set of colors C, a function $col : V \to C$, a multiset M over C, an integer k.
- **Output**: A subset $V' \subseteq V$ such that (i) $|V'| = k$, (ii) $G[V']$ is connected, and (iii) $col(V') = M$.

In the following, the motif is said colorful if M is a set (it is a multiset otherwise). Note that this problem also has application in the context of mass spectrometry [4], and may be used in social or technical networks [3,23].

Not surprisingly, the problem remains NP-complete, even under strong restrictions (when G is a bipartite graph with maximum degree 4 and M is built over two colors only [11], or when M is colorful and G is a rooted tree of depth 2 [2] or a tree of maximum degree 3 [11]). However, for general trees and multiset motifs, the problem can be solved in $\mathcal{O}(n^{2c+2})$ time, where c is the number of distinct colors in M, while being W[1]-hard for the parameter c [11]. We also point out that the problem can be solved in polynomial time if the number of colors in M is bounded and if G is of bounded treewidth [11]. It is also polynomial if G is a caterpillar [2], or if the motif is colorful and G is a tree where the colors appears at most two times. This last result is mentioned in [9] and can be retrieved by an easy transformation to a 2-SAT instance (chapter 4 of [23]).

The difficulty of this problem is counterbalanced by its fixed-parameter tractability when the parameter is k, the size of the solution [17,11,3,5,14,13] (for information about parameterized complexity, one can for example see [18]). The currently fastest algorithms in FPT for Exact Graph Motif run in $\mathcal{O}^*(2^k)$ time for the colorful case, $\mathcal{O}^*(4^k)$ time for the multiset case, and in both cases use polynomial space [13] (the $\mathcal{O}^*$ notation suppresses polynomial factors). A recent paper shows that the problem is unlikely to admit polynomial kernels, even on restricted classes of trees [2].

To deal with the high rate of noise in biological data, different variants of Exact Graph Motif have been introduced. The approach of Dondi *et al.* requires a solution with a minimum number of connected components [8], while the one of Betzler *et al.* asks for a 2-connected solution [3]. As for traditional bioinformatics problems, some colors can be inserted in a solution, or conversely, some colors of the motif can be deleted in a solution [5,8,13]. Recently, Dondi *et al.* introduced a variant when the number of substitutions between colors of the motif and colors in the solution must be minimum [9].

Following this direction, we consider in Section 2 an approximation issue when one wants to maximize the size of the solution. In Section 3, we propose an inapproximability result when one wants to minimize the number of substitutions. Finally, we present in Section 4 a new requirement concerning the connectedness of the solution with one hardness result and two fixed-parameter tractable (FPT) algorithms. Due to space constraints, most proofs are in missing.

2 Maximizing the Solution Size

To deal with the high rate of noise in the biological data, one approach allows some colors of the motif to be deleted from the solution, leading to Max Graph Motif, a problem introduced by Dondi *et al.* [8].

• **Input**: A graph $G = (V, E)$, a set of colors C, a function $col : V \to C$, a multiset M over C.
• **Output**: A subset $V' \subseteq V$ such that (i) $G[V']$ is connected, and (ii) $col(V') \subseteq M$.
• **Measure**: The size of V'.

In a natural decision form, we are also given an integer k in the input and one looks for a solution of size k (the number of deletions is thus equal to $|M| - k$). The problem is known to be in the FPT class for parameter k [8,5,13]. Concerning its approximation, Max Graph Motif is APX-hard, even when G is a tree of maximum degree 3, the motif is colorful and each color appears at most two times in G (in the same conditions, recall that the Exact Graph Motif is polynomial [9]). Moreover, there is no constant approximation ratio unless P = NP, even when G is a tree and M is colorful [8].

In the following, we answer an open question of Dondi *et al.* [8] concerning the approximation issue of the problem when G is a tree where each color occurs at most twice. To do so, we use a reduction from Max Independent Set, a problem stated as follows: Given a graph $G_I = (V_I, E_I)$, find the maximum subset $V_I' \subseteq V_I$ where there is no two nodes $u, v \in V_I'$ such that $\{u, v\} \in E_I$. Our proof proceeds in four steps. We first describe the construction of the instance $\mathcal{I}' = (G, C, col)$ for Max Graph Motif from the instance $\mathcal{I} = (G_I)$ of Max Independent Set (we consider the motif as $M = C$). We next prove that we can construct in polynomial time a solution for $\mathcal{I}'$ from a solution for $\mathcal{I}$ and, conversely, that we can construct in polynomial time a solution for $\mathcal{I}$ from a solution for $\mathcal{I}'$. Finally, we show that if there is an approximation algorithm with ratio r for Max Graph Motif, then there is an approximation algorithm with ratio r for Max Independent Set.

Before stating the reduction, consider a total order over the edges of G. We then define a function $\text{adj} : V_I \to 2^{E_I}$, giving for a node $v \in V_I$, the ordered list of edges where v is involved (thus of size $d(v)$, the degree of v). With this order, consider that $\text{adj}(v)[i]$ give the i-th edge where v is involved. From the graph $G_I = (V_I, E_I)$, we build the graph $G = (V, E)$ as follows (see also Figure 1):

- $V = \{r\} \cup \{v_i^e : 1 \leq i \leq |V_I|, e \in \text{adj}(v_i)\} \cup$
  $\{v_i^j : 1 \leq i \leq |V_I|, 1 \leq j \leq |V_I|^2\}$,
- $E = \{\{r, v_i^{\text{adj}(v_i)[1]}\} : 1 \leq i \leq |V_I|\} \cup$
  $\{\{v_i^{\text{adj}(v_i)[j]}, v_i^{\text{adj}(v_i)[j+1]}\} : 1 \leq i \leq |V_I|, 1 \leq j < d(v_i)\} \cup$
  $\{\{v_i^{\text{adj}(v_i)[d(v_i)]}, v_i^1\} : 1 \leq i \leq |V_I|\} \cup$
  $\{\{v_i^j, v_i^{j+1}\} : 1 \leq i \leq |V_I|, 1 \leq j < |V_I|^2\}$.

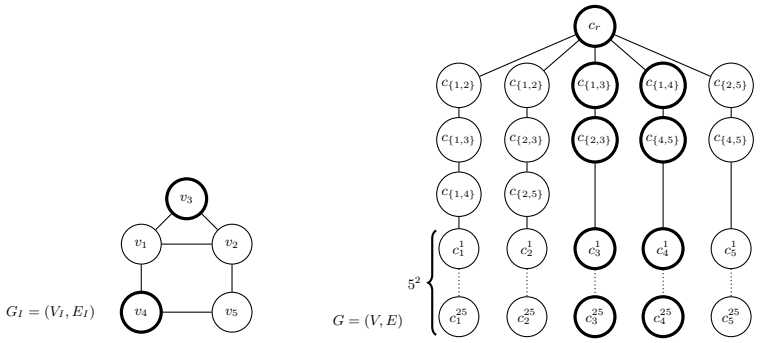


Fig. 1. Construction of G from an instance G_I of Max Independent Set. For ease, only the color of the nodes of G (not the label) are given. From a solution in G_I in bold, the solution for Max Graph Motif is given in bold in G.

Informally speaking, r is the root of G. There are $|V_I|$ paths connected to r. Each path represents a node of G_I and is of length $d(v_i) + |V_I|^2$. Observe that $|V| = 1 + 2|E_I| + |V_I|.|V_I|^2$ (there are two nodes involved in each edge, therefore $\sum_{v \in V_I} d(v) = 2|E_I|$). Let us now describe the set C of colors and the coloration function $col : V \to C$. The set of colors is $C = \{c_r\} \cup \{c_e : e \in E_I\} \cup \{c_i^j : 1 \leq i \leq |V_I|, 1 \leq j \leq |V_I|^2\}$. Considering $M = C$, the motif is colorful. Coloration of the nodes of G is done as follows: $col(r) = c_r$, $\forall e = \{v_i, v_j\} \in E_I, col(v_i^e) = col(v_j^e) = c_e$, $\forall 1 \leq i \leq |V_I|, 1 \leq j \leq |V_I|^2, col(v_i^j) = c_i^j$. In other words, for each edge $e = \{v_i, v_j\}$, a copy of the node v_i and a copy of the node v_j have the same color c_e, the one of the edge. Moreover, r and the nodes v_i^j all have different colors. In fact, nodes $\{v_i^j : 1 \leq i \leq |V_I|, 1 \leq j \leq |V_I|^2\}$ can be considered as "black boxes", which are given for free. We clearly observe that by construction, G is a tree where each color appears at most two times. Let us show how to build a solution for $\mathcal{I}'$ from a solution for $\mathcal{I}$, and *vice-versa*.

Lemma 1. *If there is a solution $V_I' \subseteq V_I$ for $\mathcal{I}$, then there is a solution $V' \subseteq V$ for $\mathcal{I}'$ such that $|V'| \geq |V_I'|.|V_I|^2$.*

Proof (Sketch). We build V' as follows: $V' = \{r\} \cup \{v_i^e, v_i^j : v_i \in V_I', e \in \text{adj}(v_i), 1 \leq j \leq |V_I|^2\}$. In other words, we add in V' the root r of the tree and all the paths corresponding to the nodes of V_I' (see also Figure 1). □

Lemma 2. *If there is a solution $V' \subseteq V$ for $\mathcal{I}'$, then there is a solution $V_I' \subseteq V_I$ for $\mathcal{I}$ such that $|V_I'| \geq \left\lceil \frac{|V'| - 2|E_I| - 1}{|V_I|^2} \right\rceil$.*

Proof (Sketch). For each $1 \leq i \leq |V_I|$, we add v_i in V_I' iff all the nodes $v_i^j, 1 \leq j \leq |V_I|^2$ and $v_i^e, e \in \text{adj}(v_i)$ are in V'. In other words, we add v_i in V_I' if the whole path corresponding to this node is in V'. □

These two lemmas lead to the main result of this section.

Proposition 1. *Unless* P = NP*, there is no approximation ratio lower than* $|V|^{\frac{1}{3}-\epsilon}$ *for* Max Graph Motif*, for any* $\epsilon > 0$*, even when the motif is colorful and* G *is a tree where each color of* C *appears at most two times.*

Proof. Suppose there is such a ratio r for Max Graph Motif. Then, there is an approximate solution V'_{APX} which, compared to the optimal solution V'_{OPT}, is of size $|V'_{APX}| \geq \frac{|V'_{OPT}|}{r}$.
With Lemma 1, $|V'_{OPT}| \geq |V'_{I_{OPT}}|.|V_I|^2$.
We supposed $|V'_{APX}| \geq \frac{|V'_{OPT}|}{r}$.
Therefore, $|V'_{APX}| \geq \frac{|V'_{I_{OPT}}|.|V_I|^2}{r}$.
With Lemma 2, $|V'_{I_{APX}}| \geq \left\lceil \frac{|V'_{APX}|-2|E_I|-1}{|V_I|^2} \right\rceil$.
Which leads to, $|V'_{I_{APX}}| \geq \left\lceil \frac{((|V'_{I_{OPT}}|.|V_I|^2)/r)-2|E_I|-1}{|V_I|^2} \right\rceil$.
Since $\frac{2|E_I|+1}{|V_I|^2} \leq 1$, $|V'_{I_{APX}}| \geq \left\lceil \frac{(|V'_{I_{OPT}}|.|V_I|^2)/r}{|V_I|^2} \right\rceil - 1 = \frac{|V'_{I_{OPT}}|}{r} - 1$.

Thereby, if there is an approximation algorithm with ratio r for Max Graph Motif, there is an approximation algorithm with ratio r for Max Independent Set. We conclude the proof by observing that $|V| = \mathcal{O}(|V_I|^3)$ and that unless P = NP, there is no ratio lower than $|V_I|^{1-\epsilon}$ for Max Independent Set, $\forall \epsilon > 0$ [24]. □

3 Minimizing the Number of Substitutions

In this section, we focus on Min Substitute Graph Motif, a problem recently introduced by Dondi *et al.* [9]. In this variant, some colors of the motif can be deleted, but the size of the solution must be equal to $|M|$. Therefore, the deleted colors must be substituted by the same number of colors.

- **Input**: A graph $G = (V, E)$, a set of colors C, a function $col : V \to C$, a multiset M over C.
- **Output**: A subset $V' \subseteq V$ such that (i) $|V'| = |M|$ and (ii) $G[V']$ is connected.
- **Measure**: The number of substitutions to get M from $col(V')$.

Dondi *et al.* [9] prove that Min Substitute Graph Motif is NP-hard, even when G is a tree of maximum degree 4 where each color occurs at most twice and the motif is colorful. On the positive side, they prove that the problem is in the FPT class when the parameter is the size of the solution.

Unfortunately, even in restrictive conditions (when G is a tree of depth 2 and the motif is colorful), we prove that there is no approximation ratio within $c \log |V|$, for a constant c. To prove such inapproximability result, we do an L-reduction from Min Set Cover [20], a problem stated as follows: Given a set $X = \{x_1, x_2, \ldots, x_{|X|}\}$ and a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}$ of subsets of X, find the minimum subset $\mathcal{S}' \subseteq \mathcal{S}$ such that every element of X belongs to at

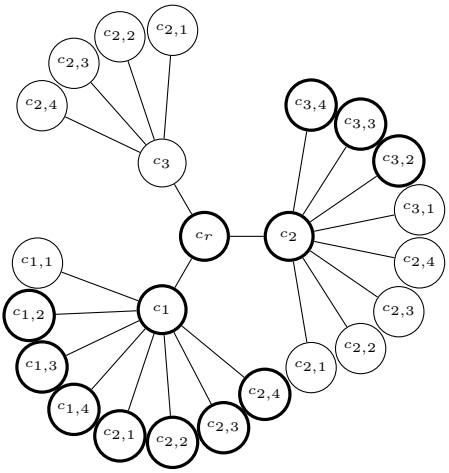


Fig. 2. Illustration of the construction of an instance of Min Substitute Graph Motif from an instance of Min Set Cover such that $X = \{x_1, x_2, x_3\}$ and $\mathcal{S} = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_2\}\}$. For ease, only the color of each node of the graph (and not the label) is given. The associated motif is $M = \{c_r\} \cup \{c_{k,t} : 1 \leq k \leq 3, 1 \leq t \leq 4\}$. A possible solution (with two substitutions) is given in bold.

least one member of $\mathcal{S}'$. We denote by $e(i, j)$ the index l such that x_l correspond to the j-th element of S_i. We first describe the polynomial construction of $\mathcal{I}' = (G, C, col, M)$, instance of Min Substitute Graph Motif, from $\mathcal{I} = (X, \mathcal{S})$, any instance of Min Set Cover. From an instance $\mathcal{I}$, let build $G = (V, E)$ as follows (see also Figure 2):

- $V = \{r\} \cup \{v_i : 1 \leq i \leq |\mathcal{S}|\} \cup$
  $\{v_{i,j,t} : 1 \leq i \leq |\mathcal{S}|, 1 \leq j \leq |S_i|, 1 \leq t \leq |\mathcal{S}| + 1\}$,
- $E = \{\{r, v_i\} : 1 \leq i \leq |\mathcal{S}|\} \cup$
  $\{\{v_i, v_{i,j,t}\} : 1 \leq i \leq |\mathcal{S}|, 1 \leq j \leq |S_i|, 1 \leq t \leq |\mathcal{S}| + 1\}$.

Informally speaking, r is the root of a tree with $|\mathcal{S}|$ children, corresponding to each subset of $\mathcal{S}$. Each child $v_i, 1 \leq i \leq |\mathcal{S}|$, got $(|\mathcal{S}| + 1)|S_i|$ children, corresponding to $|\mathcal{S}| + 1$ copies of each element of S_i. The set of colors is $C = \{c_r\} \cup \{c_i : 1 \leq i \leq |\mathcal{S}|\} \cup \{c_{k,t} : 1 \leq k \leq |X|, 1 \leq t \leq |\mathcal{S}| + 1\}$. The coloring function is such that the root has a unique color, *i.e.* $col(r) = c_r$. Each node v_i is colored with the unique color corresponding to the subset of $\mathcal{S}$, $col(v_i) = c_i, \forall 1 \leq i \leq |\mathcal{S}|$. Each node $v_{i,j,t}$ get the color of the copy of the represented element, *i.e.* $col(v_{i,j,t}) = c_{e(i,j),t}$. Finally, the motif is $M = \{c_r\} \cup \{c_{k,t} : 1 \leq k \leq |X|, 1 \leq t \leq |\mathcal{S}| + 1\}$. Observe that the colors $\{c_i : 1 \leq i \leq |\mathcal{S}|\}$ are not in the motif (which is colorful by construction). Let now show how to build a solution for $\mathcal{I}'$ from a solution for $\mathcal{I}$, and *vice-versa*.

Lemma 3. *If there is a solution $\mathcal{S}'$ for an instance $\mathcal{I}$ of* Min Set Cover*, there is a solution for the instance $\mathcal{I}'$ of* Min Substitute Graph Motif *with $|\mathcal{S}'|$ substitutions.*

Lemma 4. *From a solution for the instance $\mathcal{I}'$ for* Min Substitute Graph Motif *with at most s substitutions, there is a solution for the instance $\mathcal{I}$ for* Min Set Cover *of size at most s.*

Proof. Let $V' \subseteq V$ be a solution for $\mathcal{I}'$ such that we can obtain M from $col(V')$ with at most s substitutions. We can suppose that $s < |\mathcal{S}|+1$, otherwise, $\mathcal{S}' = \mathcal{S}$ is a solution of correct size. Solution for $\mathcal{I}$ is built as follows: $\mathcal{S}' = \{S_i : v_i \in V'\}$. If for some $1 \leq k \leq |X|$ there is no color of the set $\{c_{k,t} : 1 \leq t \leq |\mathcal{S}|+1\}$ in the solution, it means that these $|\mathcal{S}|+1$ colors have all been substituted, which is a contradiction with the supposed maximum number of s substitutions. Therefore, for each $1 \leq k \leq |X|$, there is at least one color from the set $\{c_{k,t} : 1 \leq t \leq |\mathcal{S}|+1\}$ in the solution. Thus, since the solution is connected, all elements of X are covered by $\mathcal{S}'$. Finally, the size of $\mathcal{S}'$ is bounded by s. Indeed, since their colors are not in the motif, there are at most s nodes v_i in V'. □

We can now state the main result of this section.

Proposition 2. *Unless* P = NP*, there is no polynomial approximation algorithm for* Min Substitute Graph Motif *with a ratio lower than $c \log |V|$, where c is a constant, even when the motif is colorful and G is a tree of depth 2.*

As a corollary of Proposition 2, we remark that the reduction is also an parameterized reduction. Since Min Set Cover is W[2]-hard if parameterized by the number of subsets in the solution [18], Min Substitute Graph Motif is also W[2]-hard if parameterized by the number of substitutions.

Corollary 1. Min Substitute Graph Motif *is* W[2]*-hard when parameterized by the number of substitutions.*

4 Using Modularity

In this section, we introduce a variant of Exact Graph Motif, where the connectivity constraint is replaced by modularity. After a quick recall on the modules properties, we justify this new variant. The problem stays NP-hard, however, the tools offered by the modularity allow efficient algorithms.

4.1 Definitions and Properties

In an undirected graph $G = (V, E)$, a node x *separates* two nodes u and v iff $\{x, u\} \in E$ and $\{x, v\} \notin E$. A *module* $\mathcal{M}$ of a graph G is a set of nodes not separated by any node of $V \setminus \mathcal{M}$. In other words, a module $\mathcal{M}$ is such that $\forall x \notin \mathcal{M}, \forall u, v \in \mathcal{M}, \{x, u\} \in E \Leftrightarrow \{x, v\} \in E$ [6]. The whole set of nodes V and any singleton set $\{u\}$, where $u \in V$, are the trivial modules. Before stating the definition of specific modules, let say that two modules A and B *overlap* if (i) $A \cap B \neq \emptyset$, (ii) $A \setminus B \neq \emptyset$, and (iii) $B \setminus A \neq \emptyset$. According to [6], if two modules A and B overlap, then $A \cap B$, $A \cup B$ and $(A \cup B) \setminus (A \cap B)$ are also modules. This allows the definition of *strong modules*. A module is *strong* if no

other module overlaps it, otherwise it is *weak*. Therefore, two strong modules are either included into the other, either of empty intersection. A module $\mathcal{M} \subset S$ is said *maximal* for a given set of nodes S (by default the set of nodes V) if there is no module $\mathcal{M}'$ s.t. $\mathcal{M} \subset \mathcal{M}' \subset S$. In other words, the only module which contains the maximal module $\mathcal{M}$ is S.

There are three types of modules : (i) *parallel*, when the subgraph induced by the nodes of the module is not connected (it is a parallel composition of its connected components), (ii) *series*, when the complement of the subgraph induced by the nodes of the module is not connected (it is a series composition of the connected components of its complement), or (iii) *prime*, when both the subgraph induced by the nodes of the module and its complement are connected.

The inclusion order of the maximal strong modules defines the *modular tree decomposition* $\mathcal{T}(G)$ of G, which is enough to store the whole set of strong modules. The tree $\mathcal{T}(G)$ can be recursively built by a top-down approach, where the algorithm recurs on the graph induced by the considered strong module. The root of this tree is the set of all nodes V while the leaves are the singleton sets $\{u\}$, $\forall u \in V$. Each node of $\mathcal{T}(G)$ got a label representing the type of the strong module, *parallel*, *series* or *prime*. Children of an internal node $\mathcal{M}$ are the maximal submodules of $\mathcal{M}$ (*i.e.* they are disjoints). The modular tree decomposition can be obtained with a linear time algorithm, (*e.g.* the one described in [15]). We can now introduce an essential property of $\mathcal{T}(G)$:

Theorem 1. *([6]) A module of G is either a node of $\mathcal{T}(G)$, either a union of children (of depth 1) of a series or parallel node in $\mathcal{T}(G)$.*

One can see strong modules as generators of the modules of G: the set of all modules of G can be obtained from the tree $\mathcal{T}(G)$. A crucial point to note is that there is potentially an exponential number of modules in a graph (*e.g.*, the clique K_n has 2^n modules), but the size of $\mathcal{T}(G)$ is $\mathcal{O}(n)$ (more precisely, $\mathcal{T}(G)$ has less than $2n$ nodes since there are n leaves and no node with exactly one child). Therefore, the exponential-sized family of modules of G can be represented by the linear sized tree $\mathcal{T}(G)$.

4.2 When Modules Join Graph Motif

In the following, we investigate the algorithmic issues of other topology-free definition, when replacing the connectedness demand by modularity. Following definition of Exact Graph Motif, we introduce Module Graph Motif.

- **Input**: A graph $G = (V, E)$, a set of colors C, a function $col : V \to C$, a multiset M on C of size k.
- **Output**: A subset $V' \subseteq V$ such that (i) V' is a module of G and (ii) $col(V') = M$.

This definition links the modularity demand with the motif research. The module definition implies that all the nodes in this module have a uniform relation with the set of all the other nodes outside of the module. The module nodes

are indistinguishable from the outside, they are acting similarly with the other nodes of the graph.

Authors of [1] define a biological module as a set of elements having a separable function from the rest of the graph. Similarly, authors of [19] describe a biological module as a set of some elements with an identifiable task, separable from the functions of the other biological modules. Moreover, it is shown in [7] that genes with a similar neighborhood have chances to be in a same biological process. It is thus possible that set of nodes in an algorithmic module of a graph representing a biological network have a common biological function. Also note that authors of [21] describe modules in gene regulatory networks as groups of genes which obey to the same regulations, and consequently, as groups which members cannot be distinguished from the rest of the network.

Moreover, apart of using modules in a slightly different goal (in order to predict more cleverly results of PPI), Gagneur *et al.* [12] note that modules of a graph can join biological modules, and consider modular decomposition as a general tool for biological network analysis under different representations (oriented graphs, hyper-graphs...).

However, there is no clear definition of what is (or should be) a biological module in a network [1]. We thus claim that the approach using modular decomposition is complementary to the previous definitions of biological modules (*e.g.* connected occurrences or compact occurrences).

4.3 Difficulty of the Problem

Unfortunately, Module Graph Motif is NP-hard, even under strong restrictions, *i.e.* when G is a collection of paths of size three, and when the motif is colorful. Observe that under the same conditions, Exact Graph Motif is trivially polynomial-time solvable.

Proposition 3. Module Graph Motif *is* NP*-Complete even if G is a collection of paths of size 3 and M is colorful.*

Proof (Sketch). To prove the hardness of Module Graph Motif, we propose a reduction from the NP-complete problem Exact Cover by 3-Sets (X3C) stated as follows: Given a set $X = \{x_1, \ldots, x_{3q}\}$ and a collection $\mathcal{S} = \{S_1, \ldots, S_{|\mathcal{S}|}\}$ of 3-elements subsets of X, does $\mathcal{S}$ contains a subcollection $\mathcal{S}' \subseteq S$ such that each element of X occurs in exactly one element of S'?

Let us now describe the construction of an instance $\mathcal{I}' = (G, C, col)$ of Module Graph Motif from an arbitrary instance $\mathcal{I} = (X, \mathcal{S})$ of X3C. The graph $G = (V, E)$ is built as follows: $V = \{v_i^j : 1 \leq i \leq |\mathcal{S}|, x_j \in S_i\}$, $E = \{\{v_i^1, v_i^2\} \cup \{v_i^2, v_i^3\} : 1 \leq i \leq |\mathcal{S}|\}$. Informally speaking, G is a collection of $|\mathcal{S}|$ paths with three nodes (recall that for each $1 \leq i \leq |\mathcal{S}|, |S_i| = 3$). The set of colors is $C = \{c_i : 1 \leq i \leq |X|\}$. The coloration of G is such that $col(v_i^j) = c_j$. In other words, each node get the color of the represented element of X. We also consider the colorful motif as $M = C$. □

4.4 Algorithms for the Decision Problem

Even if the problem is hard under strong restrictions, the modular decomposition tree is a useful structure to design efficient algorithms. More precisely, we show in the sequel that Module Graph Motif is in the FPT class when the parameter is the size of the solution, with a better complexity than for Exact Graph Motif when the motif is a multiset. As a corollary, we show that the problem can be solved in polynomial time if the number of colors is bounded. Moreover, Module Graph Motif is still in the FPT class if a set of colors is associated to each node of the graph.

Let us first observe that asking for a strong module instead of any module in the definition of Module Graph Motif leads to a linear algorithm. Indeed, one can just browse $\mathcal{T}(G)$ and test if the set of colors for each strong module is equal to the motif.

Let us now show an algorithm with a time complexity of $\mathcal{O}^*(2^k)$, where k is the size of the solution, for Module Graph Motif, even if the motif is a multiset. To the best of our knowledge, we do not know an algorithm with a time complexity lower than $\mathcal{O}^*(4^k)$ for Exact Graph Motif when the motif is a multiset.

Proposition 4. *There is an algorithm for* Module Graph Motif *with a time complexity of* $\mathcal{O}(2^k|V|^2)$ *and a space complexity of* $\mathcal{O}(2^k|V|)$*, where* k *is the size of the motif and of the solution.*

Proof (Sketch). Since $|M| = k$, observe that there are at most 2^k different multisets M' such that $M' \subseteq M$. We first build in polynomial time the modular tree decomposition $\mathcal{T}(G)$ from G. We repeat the following algorithm for each node $\mathcal{M}$ of $\mathcal{T}(G)$.

We start by testing if the set of the colors of $\mathcal{M}$ is exactly equal to the motif M. If it is the case, the algorithm terminates. Otherwise, if $\mathcal{M}$ is a series or parallel node, a module can be a union of its children. Given an arbitrary order on its t children, denote by $\text{Child}(\mathcal{M})[i]$ the i-th child of $\mathcal{M}$. We then delete all children $\mathcal{M}'$ of $\mathcal{M}$ such that $col(\mathcal{M}') \not\subset M$, where $col(\mathcal{M}')$ is the set of colors of the nodes of $\mathcal{M}'$. Indeed, such a child cannot be in a solution considering M. We note that the set of colors for each child correspond to a multiset $M' \subseteq M$. Since any union of children of $\mathcal{M}$ is a module of G, it is thus a potential solution. We propose to test by dynamic programming if such union corresponds to a solution for M. We build a table $D(i, M')$, for $0 \leq i \leq t$ and $M' \subseteq M$. Therefore, D has $t + 1$ lines and 2^k columns. We fill this table as follows:

$$D(0, M') = True \text{ if } M' = \{0, \ldots, 0\}, \; False \text{ otherwise,}$$
$$D(i, M') = D(i-1, M') \vee D(i-1, M' \setminus col(\text{Child}(\mathcal{M})[i])) \text{ if } i \leq t, M' \subseteq M.$$

The algorithm returns $True$ iff $D(t, M) = True$. Informally speaking, the first part of the computation of $D(i, M')$ ignores the i-th child of $\mathcal{M}$ while the second part add this child into the potential solution. □

Corollary 2. Module Graph Motif *is in* FPT *when parameterized by* $(k, |C|)$.

Proof. Note that, by definition of the motif M, for each color $c \in C$, $occ_M(c) \leq k$. Thus, the number of multisets M' such that $M' \subseteq M$ is less than $k^{|C|}$. The time complexity of the algorithm in Proposition 4 is bounded by $\mathcal{O}(k^{|C|}|V|^2)$.

This corollary is quite surprising and shows a fundamental difference with Exact Graph Motif. Indeed, recall that this problem is NP-complete, even when the motif is built over two different colors [11].

Let us now show that even when a set of colors is associated to each node of the graph, the problem is still in the FPT class. It is indeed biologically relevant to consider many functions for a same reaction in a metabolic network or to consider more than one homology for a protein in a PPI network [17,3]. A version of Exact Graph Motif with a set of colors for each graph node as been defined, and thus, we can introduce the analogous problem List-Colored Module Graph Motif.

• **Input**: A graph $G = (V, E)$, an integer k, a set of colors C, a multiset M over C, a function $col : V \to 2^C$ giving a set of colors for each node of V.
• **Output**: A subset $V' \subseteq V$ such that (i) $|V'| = k$, (ii) V' is a module of G and (iii) there is a bijection $f : V' \to M$ such that $\forall v \in V', f(v) \in col(v)$.

Proposition 5. List-Colored Module Graph Motif *is in the* FPT *class.*

4.5 Open Problems

Clearly, the noise in the biological data implies that searching exact occurrences of modules is too restrictive to consider a practical evaluation. Indeed, only one false positive or one false negative can suppress a potential solution. Adding flexibility as in variants for Exact Graph Motif seems essential. Deletions can be easily handled, but what about the insertions of colors or of nodes not in a module? It would also be interesting to know if Module Graph Motif is W[1]-hard if the parameter is the number of colors in the motif as for Exact Graph Motif, or if using modularity change the complexity class.

The complexity of the algorithm of Proposition 5 is not satisfying for practical issues. We believe that this complexity can be improved by the use of multilinear monomials detection [16].

References

1. Alm, E., Arkin, A.P.: Biological Networks. Curr. Opin. Struct. Biol. 13(2), 193–202 (2003)
2. Ambalath, A.M., Balasundaram, R., Rao H., C., Koppula, V., Misra, N., Philip, G., Ramanujan, M.S.: On the Kernelization Complexity of Colorful Motifs. In: Raman, V., Saurabh, S. (eds.) IPEC 2010. LNCS, vol. 6478, pp. 14–25. Springer, Heidelberg (2010)
3. Betzler, N., Fellows, M.R., Komusiewicz, C., Niedermeier, R.: Parameterized Algorithms and Hardness Results for Some Graph Motif Problems. In: Ferragina, P., Landau, G.M. (eds.) CPM 2008. LNCS, vol. 5029, pp. 31–43. Springer, Heidelberg (2008)

4. Böcker, S., Rasche, F., Steijger, T.: Annotating Fragmentation Patterns. In: Salzberg, S.L., Warnow, T. (eds.) WABI 2009. LNCS, vol. 5724, pp. 13–24. Springer, Heidelberg (2009)
5. Bruckner, S., Hüffner, F., Karp, R.M., Shamir, R., Sharan, R.: Topology-Free Querying of Protein Interaction Networks. J. Comput. Bio. 17(3), 237–252 (2010)
6. Chein, M., Habib, M., Maurer, M.-C.: Partitive hypergraphs. Discrete Math. 37(1), 35–50 (1981)
7. Costanzo, M., et al.: The Genetic Landscape of a Cell. Science 327(5964), 425–431 (2010)
8. Dondi, R., Fertin, G., Vialette, S.: Complexity issues in vertex-colored graph pattern matching. J. Discr. Algo. 9(1), 82–99 (2011)
9. Dondi, R., Fertin, G., Vialette, S.: Finding Approximate and Constrained Motifs in Graphs. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 388–401. Springer, Heidelberg (2011)
10. Edwards, A.M., et al.: Bridging structural biology and genomics: assessing protein interaction data with known complexes. Trends Gen. 18(10), 529–536 (2002)
11. Fellows, M.R., Fertin, G., Hermelin, D., Vialette, S.: Sharp Tractability Borderlines for Finding Connected Motifs in Vertex-Colored Graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 340–351. Springer, Heidelberg (2007)
12. Gagneur, J., Krause, R., Bouwmeester, T., Casari, G.: Modular decomposition of protein-protein interaction networks. Genome Biol. 5(8), R57 (2004)
13. Guillemot, S., Sikora, F.: Finding and counting vertex-colored subtrees. Algorithmica, 10.1007/s00453-011-9600-8
14. Guillemot, S., Sikora, F.: Finding and Counting Vertex-Colored Subtrees. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 405–416. Springer, Heidelberg (2010)
15. Habib, M., de Montgolfier, F., Paul, C.: A Simple Linear-Time Modular Decomposition Algorithm for Graphs, Using Order Extension. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 187–198. Springer, Heidelberg (2004)
16. Koutis, I., Williams, R.: Limits and Applications of Group Algebras for Parameterized Problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 653–664. Springer, Heidelberg (2009)
17. Lacroix, V., Fernandes, C.G., Sagot, M.-F.: Motif search in graphs: application to metabolic networks. IEEE/ACM T. Comput. Bi. 3(4), 360–368 (2006)
18. Niedermeier, R.: Invitation to Fixed Parameter Algorithms. Lecture Series in Mathematics and Its Applications. Oxford University Press (2006)
19. Ravasz, E., et al.: Hierarchical Organization of Modularity in Metabolic Networks. Science 297(5586), 1551–1555 (2002)
20. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: STOC, pp. 475–484. ACM (1997)
21. Segal, E., et al.: Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. Nat. Genet. 34(2), 166–176 (2003)
22. Sharan, R., Ideker, T.: Modeling cellular machinery through biological network comparison. Nat. Biotechnol. 24(4), 427–433 (2006)
23. Sikora, F.: Aspects algorithmiques de la comparaison d'éléments biologiques. PhD thesis, Université Paris-Est (2011) (in French)
24. Zuckerman, D.: Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. Theor. Comput. 3(1), 103–128 (2007)

A Characterization of Cellular Automata Generated by Idempotents on the Full Shift*

Ville Salo

University of Turku, Finland
vosalo@utu.fi

Abstract. In this article, we discuss the family of cellular automata generated by so-called idempotent cellular automata (CA G such that $G^2 = G$) on the full shift. We prove a characterization of products of idempotent CA, and show examples of CA which are not easy to directly decompose into a product of idempotents, but which are trivially seen to satisfy the conditions of the characterization. Our proof uses ideas similar to those used in the well-known Embedding Theorem and Lower Entropy Factor Theorem in symbolic dynamics. We also consider some natural decidability questions for the class of products of idempotent CA.

1 Introduction

Two famous theorems in symbolic dynamics, namely the Embedding Theorem and the Lower Entropy Factor Theorem [7], have a similar flavor. In both, we have two subshifts of finite type X and Y, such that $h(Y) > h(X)$. We then use the greater entropy of Y to encode every block of X in a suitable size range into a unique block of Y of the same length, such that the corresponding block of Y is always marked by a unique occurrence of an unbordered word w. The fact that we find sufficiently many such blocks in Y is a simple consequence of entropy.

The main problem then becomes handling the periodic parts of a point, since in a long subword of period p, the words w would need to be at most p apart. This means that the possibility of encoding does not follow from a simple entropy argument. In fact, in both theorems, the necessary and sufficient conditions include an obvious requirement for periodic points, which doesn't automatically follow.

In this article, we solve a third problem using similar argumentation. Unlike the Embedding Theorem and the Lower Entropy Factor Theorem, which are inherently about subshifts, this is a problem for cellular automata: the problem of characterizing the cellular automata F that arise as products of idempotent cellular automata (CA G such that $G^2 = G$). We only consider the case of the full shift in this paper; the case of a mixing SFT would only add some notational overhead, and we will consider this, and further extensions, in a separate paper.

* Research supported by the Academy of Finland Grant 131558

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 290–301, 2012.

It is easy to see that, apart from the trivial case of the identity CA, such a cellular automaton cannot be surjective. The higher entropy of the domain, and an obvious requirement on how F acts on periodic points, are then used to construct F as a product of idempotent CA.

The problem of characterizing the products of idempotent CA arose from its superficial similarity to the well-known open problem of characterizing the products of involutions (CA G such that $G^2 = 1$) [3]. Both problems are about the submonoid of all CA (with respect to composition) generated by a family of CA that, on their own, have very simple dynamics. In fact, just like involutions are the simplest possible type of reversible CA in the sense of generating the smallest possible nontrivial submonoids, idempotents give the simplest possible nontrivial non-surjective dynamics in the same sense. As we shall see, idempotent CA are much easier to handle than involutions, and the obvious necessary condition turns out to be sufficient.

In the process of proving the characterization, we also construct two CA which may be of interest on their own: In Lemma 6, given a non-surjective CA F, we construct a non-surjective idempotent CA E' such that $F(E'(x)) = F(x)$. This can be considered a 'CA realization' of the Garden of Eden Theorem. Also, from the Marker Theorem, we directly extract a cellular automaton M marking a 'not too dense' subset of the coordinates which is 'not too sparse' outside periodic parts of the given point. This way we see that the Marker Lemma in its full generality essentially follows from proving it for the full shift, so a uniform set of markers can be effectively constructed which works for even highly uncomputable subshifts. Lemma 4 may also be of independent interest.

We will show examples of (types of) cellular automata which are not easily decomposable into a product of idempotents, but which are trivially seen to satisfy the conditions of the characterization. Finally, we discuss decidability questions, showing that it is decidable whether a cellular automaton can be decomposed into a product of idempotents, and that many natural questions that are undecidable for one-dimensional CA stay undecidable restricted to products of idempotent CA.

2 Definitions and Useful Lemmas

For points $x \in S^{\mathbb{Z}}$, we use the term *subword* for all contents of finite, one-way infinite and bi-infinite continuous segments that occur in x. A subword u is p-periodic if $u_i = u_{i+p}$ whenever both i and $i + p$ are indices of u, and periodic if it is p-periodic for some $p > 0$.

Definition 1. *A subset $X \subset S^{\mathbb{Z}}$ is called a* subshift *if it is topologically closed in the product topology of $S^{\mathbb{Z}}$ and invariant under the left shift. This amounts to taking exactly the points $x \in S^{\mathbb{Z}}$ not containing an occurrence of a subword from a possibly infinite set of forbidden patterms. If this set of forbidden patterns can be taken to be finite, X is said to be of finite type (an* SFT*).*

In this paper, a *cellular automaton* (or *CA*) is defined as a continuous function between two subshifts X and Y which commutes with the left shifts of X and

Y. Such functions f are defined by local maps $F : S^{[-r,r]} \to S$ by $f(x)_i = F(x_{[i-r,i+r]})$. The *radius* of a CA is any r that can be used in the local map, and the *neighborhood* of a CA on the full shift is the (unique, relative to i) set of cells on which its image at i actually depends. We say F is a cellular automaton on the subshift Z if $X = Y = Z$. Note that our definition of a cellular automaton does not require the domain and codomain to be equal, in order to keep our terminology consistent. The term sliding block code is also used in symbolic dynamics [7]. We denote the identity CA defined by $G(x) = x$ by id. If X is the image of an SFT under a cellular automaton, X is said to be *sofic*.

Definition 2. *The composition, or* product*, of two CA F and G is denoted in the usual way when the range of G coincides with the domain of F: $(F \circ G)(x) = F(G(x))$. Note that $(F \circ G)$ is a cellular automaton.*

Definition 3. *By Q_n we denote the set of points of $S^{\mathbb{Z}}$ with* least *period n.*

Definition 4. *By $\mathcal{IDEMP}(X)$, we denote the set of idempotent CA on X, that is, CA $G : X \to X$ such that $G^2 = G$. When the alphabet S is obvious from context, we will also write $\mathcal{IDEMP} = \mathcal{IDEMP}(S^{\mathbb{Z}})$. Given a subshift X and a class $\mathcal{CLS}$ of cellular automata $F : X \to X$, we write $\mathcal{CLS}^*$ for the class of cellular automata $F : X \to X$ that appear as products of CA in $\mathcal{CLS}$.*

Definition 5. *For $u \in S^n$ ($x \in S^{\mathbb{Z}}$), we write $\mathcal{L}(u)$ ($\mathcal{L}(x)$) for the subwords of u (finite subwords of x). For a subshift $X \subset S^{\mathbb{Z}}$, we write $\mathcal{L}(X) = \bigcup_{x \in X} \mathcal{L}(x)$.*

Definition 6. *A set of words $V = \{v_1, \ldots, v_n\}$ is said to be* mutually unbordered *(or $v_1, \ldots, v_n$ are mutually unbordered) if for all $v_i, v_j \in V$*

$$x_{[c_1, c_1+|v_i|-1]} = v_i, x_{[c_2, c_2+|v_j|-1]} = v_j, c_1 \leq c_2 \implies$$

$$c_2 - c_1 \geq |v_i| \vee (c_1 = c_2 \wedge v_i = v_j)$$

A word v is said to be unbordered *if the set $\{v\}$ is mutually unbordered.*

Definition 7. *We say that a cellular automaton F is preinjective, if for all $x, y \in S^{\mathbb{Z}}$ such that $x \neq y$, and $x_j = y_j$ for all $|j| \geq N$ for some N, we have $F(x) \neq F(y)$.*

Definition 8. *We say that the subshift $X \subset S^{\mathbb{Z}}$ is mixing if for all $u, v \in \mathcal{L}(X)$, and for all sufficiently large n, there exists w with $|w| = n$ such that $uwv \in \mathcal{L}(X)$. It is easy to see that for a mixing SFT X, there is a uniform* mixing distance *m such that for any two words $u, v \in \mathcal{L}(X)$, and for all $n \geq m$, $uwv \in \mathcal{L}(X)$ for some w with $|w| = n$.*

Definition 9. *The kth SFT approximation of a subshift X is the SFT obtained by allowing exactly the subwords of length k that occur in X.*

We will need three classical results from the literature. First, we state the following version of the Garden of Eden theorem. This is a straightforward combination of Theorem 8.1.16 and Corollary 4.4.9 of [7].

Lemma 1 (Garden of Eden Theorem). *Let X be a mixing SFT. A cellular automaton $F : X \to X$ is preinjective if and only it is surjective.*

For the full shift, the two directions were first proved in [9] and [10]. We will need both directions of the Garden of Eden Theorem in the proof of Lemma 6.

The following is a version of Lemma 10.1.8 from [7] where instead of giving a set of cylinders F, we give a cellular automaton that, on $x \in X$, mark the cells i such that $\sigma^i(x) \in \bigcup F$ with a 1, outputting 0 on all other cells.

Lemma 2 (Marker Lemma). *[7] Let X be a shift space and let $N \geq 1$. Then there exists a cellular automaton $M : X \to \{0,1\}^{\mathbb{Z}}$ such that*

- *the distance between any two 1's in $M(x)$ is at least N, and*
- *if $M(x)_{(i-N,i+N)} = 0^{2N-1}$, then $x_{[i-N,i+N]}$ is p-periodic for some $p < N$.*

Our version of the Marker Lemma is clearly equivalent to that of [7], but makes it clearer that the marker CA for $S^{\mathbb{Z}}$ directly works for *all* subshifts of $S^{\mathbb{Z}}$, since we avoid the explicit use of cylinders, which by definition depend on the subshift X. Note that this in particular implies that a uniform set of words defining the cylinders used as markers works for every subshift $X \subset S^{\mathbb{Z}}$ whether or not X itself is in a any way accessible, and additional complexity in X may not increase the length of these words.

We need the following subset of a lemma from [2] (see also [8]).

Lemma 3 (Extension Lemma 2.4). *[2] Let T, T' and U be subshifts satisfying the following conditions:*

- *U is a mixing SFT.*
- *T' is a subshift of T.*
- *there exists a CA $F : T' \to U$.*
- *the period of any periodic point of T is divisible by the period of some periodic point of U.*

Then F can be extended to a CA $G : T \to U$ so that $G|_{T'} = F$.

By an application of the Extension Lemma to an extension of our own, we obtain a very useful lemma for idempotent CA, which simplifies our construction in Section 3.

Lemma 4. *Let the CA $F : X \to Y$ be surjective and idempotent for a subshift $X \subset S^{\mathbb{Z}}$ and a mixing subshift $Y \subset X$ containing a unary point (a point ${}^{\infty}a^{\infty}$ for $a \in S$). Then there exists an idempotent CA $G : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ such that $G|_X = F|_X$.*

Proof. It is easy to see that for any k the kth SFT approximation of a mixing subshift is mixing. Since F is idempotent, we have $F|Y = \mathrm{id}|Y$. Let r be the radius of F and let U be the $(2r+1)$th (mixing) SFT approximation of Y, which contains the unary point of Y. Note that the direct extension F' of F to $X \cup U$ is also idempotent, since points in X map to Y, and F' is the identity map on

the whole subshift U (since r is the radius of F). By the same argument, we may take F' to be a cellular automaton from $X \cup U$ to U.

We apply the Extension Lemma to $T = S^{\mathbb{Z}}$, $T' = X \cup U$, U, and the CA $F' : X \cup U \to U$. This gives us a CA $G : S^{\mathbb{Z}} \to U$ such that $G|_{X \cup U} = F'$. Since $G(x) \in U$ for all $x \in S^{\mathbb{Z}}$, and F' is the identity map on U, it follows that G is idempotent as a cellular automaton on $S^{\mathbb{Z}}$. On the other hand, $G|X = F'|X = F|X$, which concludes the proof.

Of course, we could prove a version of Lemma 4 for extensions to subshifts other than the full shift, as long as the periodic point condition of the Extension Lemma is satisfied.

3 Cellular Automata Generated by Idempotents on the Full Shift

We will prove the following theorem in this article.

Theorem 1. *$G \in \mathcal{IDEMP}^*$ if and only if*

$$\forall n : (G(Q_n) = Q_n \implies G|_{Q_n} = id|_{Q_n}) \wedge (G(S^{\mathbb{Z}}) = S^{\mathbb{Z}} \implies G = id). \quad (1)$$

It is easy to see that 'only if' holds.

Lemma 5. *Let $X \subset S^{\mathbb{Z}}$ be a subshift and let $G = G_n \circ \cdots \circ G_1$ for some $G_i \in \mathcal{IDEMP}(X)$. Then G satisfies* (1) *where we have $Q'_n = Q_n \cap X$ in place of Q_n.*

Proof. Let $G(Q'_n) = Q'_n$. Then for all G_i, also $G_i(Q'_n) = Q'_n$ since Q'_n is finite and points can only map from Q'_n to Q'_j with $j \leq n$. But $G_i \in \mathcal{IDEMP}(X)$ so G_i acts as identity on its image, in particular on Q'_n, and thus also G acts as identity on Q'_n.

Even more obviously, if $G(S^{\mathbb{Z}}) = S^{\mathbb{Z}}$ then G acts as identity everywhere.

It is not hard to show that binary xor-with-right-neighbor on the full shift satisfies the leftmost implication of (1), since no Q_n is mapped onto itself. However, it does not satisfy the rightmost implication, so the lhs does not imply the rhs. It is also easy to find a nonsurjective CA the does not satisfy the lhs.

Since we will prove the converse to Lemma 5 in the rest of this section in the case $X = S^{\mathbb{Z}}$, assume G satisfies (1). It is clear that the identity map is generated by idempotents, so we may assume G is not surjective. By Lemma 4, it is enough to show that the cellular automata F we construct are defined, and idempotent, on $Y \cup F(Y)$ where Y the image of the chain of CA constructed sofar.

We will construct $G = F \circ P \circ A \circ E$ as the product of the 4 CA

- E, the Garden of Eden CA;
- A, the Aperiodic Encoder CA;

– P, the Period Rewriter CA;
– F, the Finalizer CA.

The CA P will be a product of idempotent cellular automata, while the rest are idempotent themselves.

We will dedicate a short subsection to each of these cellular automata, and the crucial idea behind each CA is extracted into a lemma, except for the highly problem-specific F. In the case of Section 3.2, this is just the Marker Lemma.

3.1 Forbidding a Word from the Input: E

Let us start by rewriting the point so that some subword never appears, without changing the image of G.

Lemma 6. *Let $G' : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ not be surjective. Then there exists an idempotent non-surjective cellular automaton E' such that $G'(E'(x)) = G'(x)$.*

Proof. Let $Z \subsetneq S^{\mathbb{Z}}$ be the image of G'. The Garden of Eden theorem says there is a positive length word u that we can always rewrite to a different word u' with $|u| = |u'|$ without changing the image of G'. Clearly we may assume $|u| > 1$. We take one such u and take the automaton E' that rewrites an occurrence of u at $x_{[i,i+|u|-1]}$ to u' if

– u occurs exactly once in $x_{[i-2|u|+1,i+3|u|-2]}$
– rewriting u to u' does not introduce a new u overlapping the original occurrence.

Assume on the contrary that $E'^2 \neq E'$ and let $E'(x)_{[i,i+|u|-1]} = u$ for $x \in S^{\mathbb{Z}}$ such that E' rewrites this u to u'. The first condition makes sure that at most one rewriting could have happened such that the new u introduced overlaps $[i, i + |u| - 1]$. But this means that the second condition could not have been satisfied. Therefore, none of the cells have been rewritten, and necessarily $x_{[i,i+|u|-1]} = u$.

It is impossible for the cells at most $|u| - 1$ away from the occurrence of u to have changed, so the first condition was the reason u was not rewritten in the first place, and there is a nearby occurrence of u at j in x preventing this. But then, the two occurrences of u in i and j prevent each other from being rewritten in the whole orbit of the point x. This is a contradiction, since we assumed the occurrence at i is rewritten on the second step. This means E' must be idempotent.

Since $E'(^\infty aub^\infty) = E'(^\infty au'b^\infty)$ for $a \neq u_1$, $b \neq u_{|u|}$, E' is not preinjective, and the other direction of the Garden of Eden theorem says that its image is not the full shift.

We take $E = E'$, as given by Lemma 6 for $G' = G$, as our first idempotent CA. Let $v \notin \mathcal{L}(E(S^{\mathbb{Z}})) \cup \mathcal{L}(G(S^{\mathbb{Z}}))$ and let $Y = \{x \mid v \notin \mathcal{L}(x)\}$. We choose three mutually unbordered words w, w_0, w_1 all containing a single copy of v such that v can only overlap w, w_0 or w_1 at its unique occurrence within it. Further, we may assume $Y' = \{x \in S^{\mathbb{Z}} \mid w \notin \mathcal{L}(x)\}$ is mixing.

3.2 Encoding Aperiodic Parts and Memorizing Periodic Parts: A

Next, we construct the CA A that, when started from a point not containing the word v, marks the borders of long enough periodic subwords (with small enough period) memorizing the repeated pattern, and encodes the aperiodic parts by occurrences of v. For this, we need a suitable definition for 'long enough periodic subword' and 'small enough period'.

Let m be large enough that

$$|\{wuw \mid u \in S^{n-2|w|} \cap \mathcal{L}(Y')\}| > |\{u \in \mathcal{L}(Y) \mid |u| = n\}| \tag{2}$$

for all $n \geq m$. This is possible by a standard entropy argument since Y' is a mixing SFT and $Y \subsetneq Y'$. Note that since w is unbordered, w occurs only twice in wuw on the LHS. Let k be such that in a word of length k, no two distinct periods $p_i, p_j \leq m$ can occur.

Let $y =\in S^{\mathbb{Z}}$, and let M be given by the Marker Lemma for the full shift and $N = m + 1$ having radius r. For now, let $r' > 0$ be arbitrary (to be specified later). We construct a shift-commuting function A as follows, applying the rules top-down:

- If v occurs in $y_{[i-r',i+r']}$, the cell i is not rewritten.
- If $M(y)_{[i-1,i+2(|w_0|+m+|w_1|)+k-1]} \in 10^*$, the word $y_{[i,i+2(|w_0|+m+|w_1|)+k-1]}$ has a unique period $p \leq m$ by the Marker Lemma and the choice of k, and A sandwiches $t = y_{[i,i+p-1]}$ between w_0 and w_1 rewriting $y_{[i,i+|w_0|+p+|w_1|-1]}$ by $w_0 t w_1$.
- If $M(y)_{[i-2(|w_0|+m+|w_1|)-k+1,i+1]} \in 0^*1$, the word $y_{[i-2(|w_0|+m+|w_1|)-k+1,i]}$ has a unique period $p \leq m$ by the Marker Lemma and the choice of k, and A sandwiches $t = y_{[i-p+1,i]}$ between w_1 and w_0 rewriting $y_{[i-|w_1|-p-|w_0|+1,i]}$ by $w_1 t w_0$.
- If $M(y)_{[i,i+n+1]} = 10^n 1$ for $n \leq 2(|w_0| + m + |w_1|) + k - 2$, A injects $y_{[i,i+n]}$ into a word wuw where u does not contain w.

The last property is possible by the fact two 1's are at least $m + 1$ apart by the Marker Lemma.

We define the aperiodic subwords, the *AS*, of a point $A(E(x))$ as the maximal subwords of the form $wu_1wwu_2w \cdots wu_nw$ (an AS is, formally, a pair containing a word and the index at which it occurs in $A(E(x))$). We define the period bordering subwords, the *PBS*, as the subwords $w_j t w_{1-j}$ (again, also remembering the location). A PBS of the form $w_0 t w_1$ is called a *left border*, and a PBS of the form $w_1 t w_0$ is called a *right border*. Finally, we define the long periodic subwords, the *LPS*, as the rest of the maximal subwords not intersecting AS or PBS.

For a sufficiently large choice of r', the restriction $A : E(S^{\mathbb{Z}}) \cup A(E(S^{\mathbb{Z}})) \to A(E(S^{\mathbb{Z}}))$ is an idempotent CA: Consider a rewriting that happens on the second step at i. This i must be in an LPS if r' is chosen large enough. Also, clearly for cells i deep enough (at least $r + m$) inside a p-periodic subword with $p \leq m$, M marks no cells with a 1. It then clear that a large enough choice of r' implies the idempotency of A.

Note that, since the length of a minimal wuw-pattern is bounded, a CA can determine which type of subword i belongs to, that is, there exists a cellular automaton $T : S^{\mathbb{Z}} \to \{(\text{AS}), (\text{PBS}), (\text{LPS})\}^{\mathbb{Z}}$ such that $T(A(E(x)))_i = \mathcal{T}$ if and only if i is in a subword of type $\mathcal{T}$.

We illustrate the structure of a point $(A \circ E)(x)$ in Fig. 1.

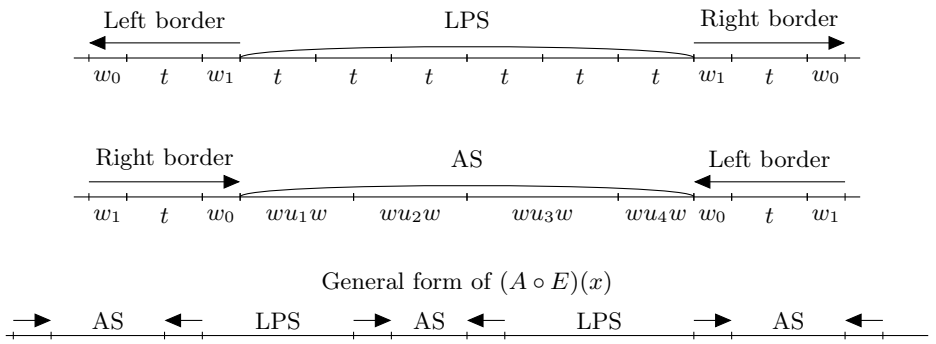


Fig. 1. An LPS, an AS, and the general structure of a point, respectively, after $A \circ E$ has been applied. Note that in reality the bordermost copies of t in an LPS are usually cut off (unlike in the figure), and the two t sandwitched between w_i are usually not the same, but rotated versions (conjugates) of each other.

3.3 Periodic Subwords of Small Enough Period: P

Now that LPS subwords can be detected by a CA in $(A \circ E)(x)$, we can deal with LPS separately from the rest of the point. So, let us construct a CA $P' \in \mathcal{IDEMP}^*$ which behaves like G on all points with period less than or equal to m. We will then modify P' to obtain the desired CA P such that $P \circ A \circ E$ writes the LPS exactly the same way as G would have rewritten the corresponding periodic point (while leaving the original periodic pattern t in the period borders $w_j t w_{1-j}$).

We start with the following lemma, which contains all the essential ideas needed in the construction of P'.

Lemma 7. *Let X be a finite set and let $f : X \to X$ not be surjective. Then there exist idempotent functions f_i such that $f = f_n \circ \cdots \circ f_1$.*

Proof. First, choose a preimage $g(a) \in f^{-1}(a)$ for all $a \in f(X)$. Then, construct a sequence of idempotent functions g_i that each move a single element $a \in X$ to $g(f(a))$, and leave everything else fixed. Next, move $g(f(X))$ onto $f(X)$ with another product of functions h_i. Finally, decompose the permutation of $f(X)$ moving every element $a \in X$ to its final position $f(a)$, into 2-cycles. Each 2-cycle can be implemented using two idempotent functions k_i and an element $b \in X - f(X)$. Letting $f = \prod_i k_i \circ \prod_i h_i \circ \prod_i g_i$ completes the construction.

Lemma 8. *Let m' be arbitrary, and let Y' be a subshift such that $G|_{Y'}$ satisfies* (1)*. Then there exists*

$$P' \in \mathcal{IDEMP}(Y')^*$$

such that P' acts as G on all points with period less than or equal to m'.

Proof. There exists k' such that by looking k' cells in each direction, we can uniquely identify the period of the point. We build P' as the product $P'_{m'} \circ \cdots \circ P'_1$ where each P'_i takes care of points with period i. If $G(Q_i \cap Y') = Q_i \cap Y'$ then P'_i is just the identity. All points that map to a point of smaller period simply map directly to that point. This is safe because of the order in which we handle the different periods, since the period of a point cannot be increased by a cellular automaton.

We deal with other points similarly to Lemma 7, simply shuffling everything in place with a product of idempotents. For this, note that $Q_i \cap Y'$ are partitioned into equivalence classes of size i by the shift, and that an equivalence class either maps to a set of points with smaller period or onto some equivalence class, possibly shifted. This means that the construction in Lemma 7 can be used on equivalence classes: In the terminology of Lemma 7, the functions g_i are composed with a suitable power of the shift, and finally, additional cellular automata l_i are used to shift the images of all points to their final image (again using a point outside of $f(X)$).

Let the CA $P' = P_h \circ \cdots \circ P_1$ be given by Lemma 8 for $m' = m$ and $Y' = \{x \in S^{\mathbb{Z}} \mid w \notin \mathcal{L}(x)\}$, where each P_i is idempotent. For this, note that if G satisfies (1) on the full shift, the equation is also satisfied on Y' (since v does not occur in the image of G and v is unbordered and contained in w).

To extend P' to P, we must make each P_i identify whether the cell being rewritten is part of an LPS. This is complicated by the fact that the intermediate CA P_i may have v in their image. However, since w does not occur in the images of the P_i, AS subwords, and thus all types of subwords, are still easy to locate, and the CA T can be extended for this case. If i is not in an LPS, the cell is not rewritten. Otherwise, the cell is rewritten as P' would have, if the periodic pattern were repeated infinitely in both directions. That is, the bordermost w_1, if seen, is thought of as a repeater, repeating whichever periodic pattern occurs inside the LPS, see Fig. 2. This is possible, since at least k cells are left between the period borders $w_j t w_{1-j}$, and the repeated pattern can be uniquely determined. This concludes the construction of P. Note that, as mentioned above, it is enough that the intermediate CA are idempotent on the image Z of the previous chain of CA and their own image from Z by Lemma 4.

The only difference in form between $(P \circ A \circ E)(x)$ and $(A \circ E)(x)$ is that the repeating subword of an LPS may have changed, see Fig. 3.

3.4 The Final Touch: F

Let l be such that if $x_{[i-l,i+l]}$ does not contain v for $x \in S^{\mathbb{Z}}$, then $(P \circ A \circ E)(x)_i = G(x)_i$. For instance, $l = \max(|w_0| + |w_1|) + r$ has this property where r is the

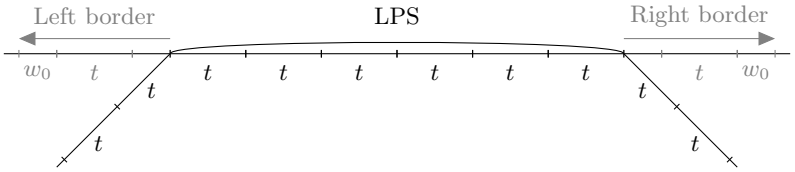


Fig. 2. An LPS as seen by P' in $(A \circ E)(x)$

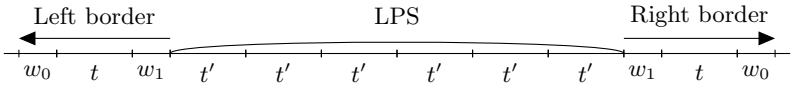


Fig. 3. An LPS after applying P to $(A \circ E)(x)$

radius of P. All we need to do is rewrite the rest of x as G would have, with a CA F. We ensure that F is idempotent by only rewriting i such that $[i-l, i+l]$ contains v, since $G(x)$ cannot contain a copy of v. But it is easy to deduce the original contents of any cell j that G might use when rewriting such a cell i:

- in an AS, between two w, the original contents are given by simply decoding wuw.
- shallow enough inside an LPS, or in a PBS, the t in $w_j t w_{1-j}$ gives the original periodic pattern repeated in x.

Now,

$$G = F \circ P \circ A \circ E$$

concludes the proof of Theorem 1.

4 Examples and Decidability Questions

4.1 Examples

It is now easy to see that while idempotent CA are in some sense trivial, their products can have complicated behavior.

We say that a CA is *nilpotent* if $\exists q, n : \forall x \in S^{\mathbb{Z}} : G^n(x) = {}^{\infty}q^{\infty}$, and we say the CA F has a *spreading state* q if q spreads to i whenever F sees q in the neighborhood of i.

Proposition 1. *If the CA F has a spreading state, has neighborhood and alphabet size at least* 2*, and satisfies* (1) *on Q_1, then $F \in \mathcal{IDEMP}^*$.*

Proof. Such a CA cannot be preinjective, and thus not surjective either, so the rightmost condition of Theorem 1 is satisfied. Also, no Q_n where $n > 1$ maps to itself.

Proposition 2. *If a non-surjective CA F has a single spatially and temporally periodic point, then $F \in \mathcal{IDEMP}^*$.*

By considering north-west deterministic tilings, we find a non-nilpotent cellular automaton on the full shift having a spreading state such that all periodic configurations eventually evolve into the all zero configuration [6]. Such CA are rather nontrivial to construct, implying that $\mathcal{IDEMP}^*$ is quite a large class.

Note that an idempotent CA is simply an eventually periodic automaton with period 1 and threshold 1. We say that a CA G is eventually idempotent if the period is 1, but the threshold need not be, that is, $G^{n+1} = G^n$ for some n. Let us show that such CA are products of idempotent CA.

Proposition 3. *If $G^{n+1} = G^n$, then $G \in \mathcal{IDEMP}^*$*

Proof. The proof of Lemma 5 can easily be modified for such CA: If $G(Q_n) = Q_n$ and G is composed of eventually idempotent CA G_i, each G_i must also map $G_i(Q_n) = Q_n$, and thus $G_i^n(Q_n) = Q_n$. From this, it follows that for all $x \in Q_n$, we have $G_i(x) = G_i(G_i^n(y)) = G_i^n(y) = x$ for some $y \in Q_n$. The right hand side of (1) follows similarly.

Corollary 1. *If G is a product of eventually periodic CA, then $G \in \mathcal{IDEMP}^*$.*

Corollary 2. *Any nilpotent CA F is in $\mathcal{IDEMP}^*$.*

This means that we have exactly characterized the products of eventually periodic CA with period 1 and an arbitrary threshold. As we mentioned in the Introduction, the case of period 2 and threshold 0 is still open.

4.2 Decidability Questions

Although we have complicated examples of CA in $\mathcal{IDEMP}^*$, the problem of whether a CA is in this class is simple to solve using our characterization.

Theorem 2. *It is decidable whether the CA F is in $\mathcal{IDEMP}^*$.*

Proof. Obviously, F being in $\mathcal{IDEMP}^*$ is semi-decidable. On the other hand, if F is not in $\mathcal{IDEMP}^*$, it does not satisfy the characterization of Theorem 1. If F does not satisfy the condition $F(S^{\mathbb{Z}}) = S^{\mathbb{Z}} \implies F = \mathrm{id}$, the cellular automaton is surjective but not equal to the identity, and since surjectivity and not being equivalent to the identity CA are both semidecidable [1] [6], a semialgorithm can detect this. If the condition $\forall n : (F(Q_n) = Q_n \implies F|_{Q_n} = \mathrm{id}|_{Q_n})$ is not satisfied, there exists n such that $F(Q_n) = Q_n$, but $F|_{Q_n} \neq \mathrm{id}|_{Q_n}$, which is easily found by enumerating the sets Q_n.

However, once restricted to CA in $\mathcal{IDEMP}^*$, we find many undecidable problems, of which we list a few. In [5], it is shown that nilpotency of cellular automata with a spreading state is undecidable. From this and Proposition 1, we obtain the following.

Theorem 3. *It is undecidable whether $F \in \mathcal{IDEMP}^*$ is nilpotent.*

By attaching a full shift (with shift dynamics) to the state set so that the spreading state also zeroes cells of the full shift, we obtain that computation of entropy up to error $\epsilon > 0$ is uncomputable even for CA with a spreading state [4]. In particular, we obtain that this is also undecidable for CA in $\mathcal{IDEMP}^*$.

Theorem 4. *Approximating the entropy of $F \in \mathcal{IDEMP}^*$ up to error ϵ is uncomputable for all $\epsilon > 0$.*

Acknowledgements. I would like to thank Ilkka Törmä for his idea of also discussing eventually idempotent cellular automata, and for his useful comments on an early version of this article. I would also like to thank Jarkko Kari for pointing out an error in the examples section.

References

1. Amoroso, S., Patt, Y.N.: Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. Journal of Computer and System Sciences 6(5), 448–464 (1972)
2. Boyle, M.: Lower entropy factors of sofic systems. Ergodic Theory Dynam. Systems 3(4), 541–557 (1983)
3. Boyle, M., Lind, D., Rudolph, D.: The automorphism group of a shift of finite type. Trans. Amer. Math. Soc. 306(1), 71–114 (1988)
4. Hurd, L.P., Kari, J., Culik, K.: he topological entropy of cellular automata is uncomputable. Ergodic Theory Dynam. Systems 12(2), 255–265 (1992)
5. Kari, J.: The nilpotency problem of one-dimensional cellular automata. SIAM J. Comput. 21(3), 571–586 (1992)
6. Kari, J.: Theory of cellular automata: a survey. Theor. Comput. Sci. 334, 3–33 (2005)
7. Lind, D., Marcus, B.: An introduction to symbolic dynamics and coding. Cambridge University Press, Cambridge (1995)
8. Maass, R.: On the sofic limit sets of cellular automata. Ergodic Theory and Dynamical Systems 15 (1995)
9. Moore, E.F.: Machine models of self-reproduction. In: Proc. Symp. Applied Mathematics, vol. 14, p. 187–203 (1962)
10. Myhill, J.: The converse of moore's garden-of-eden theorem. Proceedings of the American Mathematical Society 14, 685–686 (1963)

Constructing Polynomials for Functions over Residue Rings Modulo a Composite Number in Linear Time

Svetlana N. Selezneva

Department of Computational Mathematics and Cybernetics
Moscow State University
selezn@cs.msu.su

Abstract. We show how to check in linear time if a function $f : \mathbb{Z}_k^n \to \mathbb{Z}_k$, where $k = p^m$, p is a prime number, and $m \geq 2$, specified by its values, can be represented by a polinomial in the ring $\mathbb{Z}_k[x_1, \ldots, x_n]$. If so, our algorithm also constructs (in linear time) its canonical polynomial representation. We also show how to extend our techniques (with linear time) to the cases of an arbitrary composite number k.

More precisely, we prove that the circuit-size complexity of solving the problem, if a given function $f : \mathbb{Z}_k^n \to \mathbb{Z}_k$, where k is a fixed composite number, specified by its values, is represented by a polynomial in the ring $\mathbb{Z}_k[x_1, \ldots, x_n]$ and, if so, finding its polynomial, is linear.

1 Introduction

Polynomial is one of the most suitable forms to represent finite-valued functions since it admits the application of a wide range of algebraic techniques for symbolic manipulations with functions. It is known that each function over the ring of residues modulo k can be represented by a polynomial over this ring iff this ring is a field, i.e. iff k is a prime number.

When function $f : \mathbb{Z}_k^n \to \mathbb{Z}_k$, where k is a prime number, is specified by the vector of its $N = k^n$ values, then its polynomial can be constructed in time $O(N \log N)$.

In this paper we study the problem of checking the existence of polynomial representation and, if so, finding it (for short, PRP, polynomial representation problem) for functions over a ring of residues modulo k in the case when k is a composite integer. Some criterium for the existence of polynomial representation for functions over residue rings modulo a composite number k was obtained in [2,4,6,7,8]. In [8] D.G. Meshchanov considered the case of $k = p^m$, where p is a prime number, and $m \geq 2$, and proposed an algorithm which solves PRP in time $O(N \log^m N)$ for functions $f : \mathbb{Z}_k^n \to \mathbb{Z}_k$, specified by the vector of their $N = k^n$ values.

The main contribution of this paper is a new, more efficient solution to PRP. We introduce a new concept of canonical form for polynomial representations of functions over the ring $\mathbb{Z}_k$ [9]. We propose an algorithm that checks in time $O(N)$ if a function $f : \mathbb{Z}_k^n \to \mathbb{Z}_k$, where $k = p^m$, p is a prime number, $m \geq 2$,

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 302–313, 2012.

specified by the vector of its $N = k^n$ values, can be represented by a polynomial in the ring $\mathbb{Z}_k[x_1, \ldots, x_n]$. If so, this algorithm also conctructs the canonical polynomial that represents f. It is just a matter of a simple algebraic technique to extend our algorithm in such a way that it solves in linear time the same problem for an arbitrary composite number k. We use circuits in some basis of gates as a model of computation.

The paper is organized as follows. In Section 2 we introduce the notation and the basic concepts of finite-valued functions and polinomials. In Section 3 we define a characteristics of monomials and study its properties. In Section 4 we describe a new canonical form of polinomial functions over the ring $\mathbb{Z}_k$ where $k = p^m$, p is a prime number, and $m \geq 2$. In Section 5 we prove a criterium of polynomiality for the function over the ring $\mathbb{Z}_k$ where $k = p^m$, p is a prime number, and $m \geq 2$. Based on this criterium we construct a checking algorithm. In the end we adduce examples.

2 Concepts

In the beginning we arrange notations and agreements. We think, it is possible to use some notations from [5].

We use capital letters F, G, H, etc. to design elements of $\mathbb{Z}[x_1, \ldots, x_n]$, the polynomial ring in $x_1, \ldots, x_n$ over $\mathbb{Z}$, the ring on integers. An equality $F = G$ or $F[x_1, \ldots, x_n] = G[x_1, \ldots, x_n]$ means that F and G are equal as elements of $\mathbb{Z}[x_1, \ldots, x_n]$, i.e. their corresponding coefficients are equal.

Any polynomial in $\mathbb{Z}[x_1, \ldots, x_n]$ gives rise to a polynomial in $\mathbb{Z}_k[x_1, \ldots, x_n]$, where $k \in \mathbb{Z}$, and we use the same letter for this. Then the congruence $F \equiv G (\mathrm{mod}\ k)$ or $F[x_1, \ldots, x_n] \equiv G[x_1, \ldots, x_n] (\mathrm{mod}\ k)$ means that F and G are equal as elements of $\mathbb{Z}_k[x_1, \ldots, x_n]$,i.e. their corresponding coefficients are congruent $(\mathrm{mod}\ k)$.

We use small letters f, g, h, ets. to design functions from $\mathbb{Z}_k^n$ to $\mathbb{Z}$. Each such function gives rise to a function from $\mathbb{Z}_k^n$ to $\mathbb{Z}_k$, which we designate by the same letter. An equality $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)$ denotes equality between the integers $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$, while $f(x_1, \ldots, x_n) \equiv g(x_1, \ldots, x_n) (\mathrm{mod}\ k)$ denotes congruence between them. Then $f \equiv g (\mathrm{mod}\ k)$ means that f and g are equal as functions from $\mathbb{Z}_k^n$ to $\mathbb{Z}_k$, i.e. $f(a_1, \ldots, a_n) \equiv g(a_1, \ldots, a_n) (\mathrm{mod}\ k)$ for all $(a_1, \ldots, a_n) \in \mathbb{Z}_k^n$. The set of functions from $\mathbb{Z}_k^n$ to $\mathbb{Z}_k$ is denoted by P_k^n, and $P_k = \bigcup_{n=0}^{\infty} P_k^n$.

Given a polynomial $F(x_1, \ldots, x_n)$ in $\mathbb{Z}[x_1, \ldots, x_n]$ or in $\mathbb{Z}_k[x_1, \ldots, x_n]$, we can define a function $f : \mathbb{Z}_k^n \to \mathbb{Z}_k$ by setting $f(a_1, \ldots, a_n) \equiv F(a_1, \ldots, a_n) (\mathrm{mod}\ k)$ for all $(a_1, \ldots, a_n) \in \mathbb{Z}_k^n$. Such function is called *a polynomial* function $(\mathrm{mod}\ k)$, and we say that f *is determined by*, or *corresponds to*, F and that F *represents* f. Henceforth, we shall always use corresponding large and small letters in this way. Clearly $F \equiv G (\mathrm{mod}\ k)$ implies $f \equiv g (\mathrm{mod}\ k)$, but not conversely. The set of polynomial functions $(\mathrm{mod}\ k)$ in n variables is denoted by Pol_k^n, and $Pol_k = \bigcup_{n=0}^{\infty} Pol_k^n$.

A length $\mathrm{len}(F)$ of a polynomial F in $\mathbb{Z}_k[x_1, \ldots, x_n]$ is the number of non-zero (mod k) summands in it. The polynomial without non-zero summands we call *the empty* polynomial and denote by 0. It represents the zero function which we also designate by 0. *A degree* $\deg(F)$ of a polynomial F in $\mathbb{Z}_k[x_1, \ldots, x_n]$ is the maximum of the degrees of its non-zero (mod k) summands. Assume that the degree of 0 is zero.

Let $\mathbb{Z}_+$ be the set of non-negative integers. The partial order $\leq$ on the set $\mathbb{Z}_+^n$, $n \geq 1$, is defined as follows.

Let $\sigma = (s_1, \ldots, s_n) \in \mathbb{Z}_+^n$ and $\tau = (t_1, \ldots, t_n) \in \mathbb{Z}_+^n$. Then

$$\sigma \leq \tau, \text{ if } s_1 \leq t_1, \ldots, s_n \leq t_n.$$

We introduce the corresponding partial order on the set of monomials in $x_1, \ldots, x_n$.

For $\sigma = (s_1, \ldots, s_n) \in \mathbb{Z}_+^n$ we denote by X_σ the monomial $\prod_{i=1}^{n} x_i^{s_i}$. Given a pair of monomials X_1 and X_2 we assume

$$X_1 \leq X_2, \text{ if } X_1 = X_\sigma,\ X_2 = X_\tau, \text{and } \sigma \leq \tau.$$

3 On Some Characteristics of Monomials

Now we introduce a concept.

Definition 1. *Let p be a prime number.*

A composite characteristics (relative to p) $cd(X)$ of monomial $X = x^s$ is the integer number M, $M \geq 0$, such that

1) there exists such polynomial $G(x) = x^s + a_{s-1}x^{s-1} + \ldots + a_1 x \in \mathbb{Z}[x]$ that $g(x) \equiv 0 (\mathrm{mod}\ p^M)$;

2) for each polynomial $G(x) = b_s x^s + b_{s-1}x^{s-1} + \ldots + b_1 x \in \mathbb{Z}[x]$ if $g(x) \equiv 0 (\mathrm{mod}\ p^{M+1})$ then the coefficient b_s is divided by p.

□

We must say that this concept is suitable for our purposes. But it reinterprets the results of I. Niven & L.J. Warren [1] and D. Singmaster [5]. According to these results the value of $cd(x^s)$ is M where p^M is the highest power of p that divides $s!$, i.e.

$$p^M \mid s!,\ p^{M+1} \nmid s!$$

From this it is easy to extract the next Lemma 1.

Lemma 1. *Let p be a prime number. If $cd(x^s) < m$ (relative to p), where $m \geq 1$, then $s < mp$.*

From the results of I. Niven & L.J. Warren [1] it follows that if

$$G_s(x) = x(x-1)(x-2)\ldots(x-s+1)$$

then $g_s(x) \equiv 0 (\mathrm{mod}\ p^{cd(x^s)})$. Note that $\deg(G_s) = s$. We can conclude from the above-mentioned that if $cd(x^s) < m$ (relative to p) then $\mathrm{len}(G_s) \leq mp$.

Now we extend the concept of a composite characteristics to multivariable monomials.

Definition 2. *Let p be a prime number.*

A composite characteristics (relative to p) $cd(X)$ of monomial $X = x_1^{s_1} \cdot \ldots \cdot x_n^{s_n}$, where $s_1, \ldots, s_n \geq 0$, is the integer number $cd(X) = \sum_{i=1}^{n} cd(x_i^{s_i})$. □

Definition 3. *Let $\sigma = (s_1, \ldots, s_n) \in \mathbb{Z}_+^n$, $X_\sigma = \prod_{i=1}^{n} x_i^{s_i}$.*

We introduce the polynomial $G_\sigma(x_1, \ldots, x_n)$ as $G_\sigma(x_1, \ldots, x_n) = \prod_{i=1}^{n} G_{s_i}(x_i)$, where

$$G_{s_i}(x_i) = x_i(x_i-1)(x_i-2)\ldots(x_i-s+1),\ g_s \equiv 0(\bmod\ p^{cd(x_i^{s_i})}),\ \text{if } cd(x_i^{s_i}) \geq 1;$$

$$G_{s_i}(x_i) = x_i^{s_i},\ g_s \equiv 0(\bmod\ p^{cd(x_i^{s_i})}),\ \text{if } cd(x_i^{s_i}) = 0.$$

□

Note that

$$g_\sigma(x_1, \ldots, x_n) \equiv 0(\bmod\ p^{cd(X_\sigma)}),$$

and for each number m, $m \geq cd(X_\sigma)$,

$$p^{m-cd(X_\sigma)} g_\sigma(x_1, \ldots, x_n) \equiv 0(\bmod\ p^m).$$

Note also that all summands in the polynomial G_σ are less than or equal to X_σ in the partial order of monomials.

Lemma 2. *Let p be a prime number, $m \geq 1$, $\sigma = (s_1, \ldots, s_n) \in \mathbb{Z}_+^n$, and $cd(X_\sigma) < m$. Then the lenght of the polynomial $G_\sigma(x_1, \ldots, x_n)$ is less than $(mp)^m$.*

Proof. Consider the polynomial $G_\sigma(x_1, \ldots, x_n) = \prod_{i=1}^{n} G_{s_i}(x_i)$, where $g_{s_i}(x_i) = 0(\bmod\ p^{cd(x_i^{s_i})})$.

Let $cd(x_i^{s_i}) = m_i \geq 0$, $i = 1, \ldots, n$.

If $m_i = 0$ then $g_{s_i}(x_i) = x_i^{s_i}$, and the length of the polynomial G_{s_i} is 1.

If $m_i \geq 1$ then, since the degree of the polynomial g_{s_i} is s_i, the length of the polynomial G_{s_i} is less than or equal to s_i ($g_{s_i}(0) \equiv 0$). By Lemma 1 $s_i < mp$.

By conditions $\sum_{i=1}^{n} m_i < m$. Hence, the number of non-zero m_i is less than m.

Thus, the length of the polynomial G_σ is less than $(mp)^m$.

4 Canonical form of Polynomials

Now we formulate the Lemmas and Theorems which are used later.

The results of the next Lemma 3 and Theorem 1 can be found in [1,3,5]. We formulate them in our terms.

Lemma 3. [1,3,5] *Let p be a prime number, $m \geq 1$, and*

$$G(x) = a_s x^s + a_{s-1} x^{s-1} + \ldots + a_1 x \in \mathbb{Z}[x].$$

If $g(x) \equiv 0(\mod p^m)$, and $q = m - cd(x^s) \geq 0$, then all the coefficients $a_s, \ldots, a_1$ are divided by p^q.

Theorem 1. [3,5] *Let $k = p^m$, where p is a prime number, $m \geq 1$.*

Each function $f(x) \in Pol_k$ is uniquely represented by a polynomial of the form

$$F(x) = a_s x^s + a_{s-1} x^{s-1} + \ldots + a_1 x + a_0 \in \mathbb{Z}[x],$$

where $cd(x^s) < m$ and $0 \leq a_i < p^{m-cd(x^i)}$, $i = 0, 1, \ldots, s$.

The generalizations of these results on multivariable case (the next Lemma 4 and Theorem 2) are obtained and proved by the author in [9].

Lemma 4. [9] *Let p be a prime number, $m \geq 1$, and*

$$G(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n].$$

If $g(x_1, \ldots, x_n) \equiv 0(\mod p^m)$, X is a maximal monomial of non-zero summands in the polynomial G, and $q = m - cd(X) \geq 0$, then the coefficient a of the monomial X is divided by p^q.

Theorem 2. [9] *Let $k = p^m$, where p is a prime number, $m \geq 1$.*

Each function $f(x_1, \ldots, x_n) \in Pol_k$ is uniquely represented by a polynomial of the form

$$F(x_1, \ldots, x_n) = \sum_{i=1}^{l} a_i \cdot X_i \in \mathbb{Z}[x_1, \ldots, x_n],$$

where $cd(X_i) < m$ and $0 \leq a_i < p^{m-cd(X_i)}$, $i = 1, \ldots, l$.

A polynomial F in the form of Therem 2 is called the *canonical polynomial* of a polynomial function $f \in Pol_k$.

Let $k = p^m$, where p is a prime number, $m \geq 1$. We must say that from the results in [1,5,9] it is known how for each $m \geq 1$ we can find all monomials X such as $cd(X) < m$ (relative to p). But in our case for the simplicity we only approximate the set of such monomials.

For $r \in \mathbb{Z}_+$ we assume that $\mathbb{Z}_{[0;r]}$ denotes the set of integers from 0 to r. Then, acording to Theorem 2, there exist such coefficients $c_f(\sigma) \in \mathbb{Z}_k$, $\sigma \in \mathbb{Z}^n_{[0;r]}$ for some $r \in \mathbb{Z}_+$, that the polynomial

$$F(x_1, \ldots, x_n) = \sum_{\sigma \in \mathbb{Z}^n_{[0;r]}} c_f(\sigma) X_\sigma \in \mathbb{Z}_k[x_1, \ldots, x_n]$$

is canonical to a polynomial function $f \in Pol_k$. By Lemma 1 $r < mp$, and by the fact $mp \leq p^m$ at $p \geq 2$ we obtain $r < k$.

For the simplicity, without the loss of rigour, we sometimes write X_σ, where $\sigma = (s_1, \ldots, s_n) \in \mathbb{Z}^n_k$, for a monomial $X_{\sigma'}$, $\sigma' = (s'_1, \ldots, s'_n) \in \mathbb{Z}^n_{[0;k-1]}$, meaning that $s_1 \equiv s'_1(\mod k), \ldots, s_n \equiv s'_n(\mod k)$. Note that for every $\sigma' \in \mathbb{Z}^n_{[0;k-1]}$ there exists a unique such $\sigma \in \mathbb{Z}^n_k$.

5 Checking Algorithm of Polynomiality for Functions

Let $k = p^m$, where p is a prime number, $m \geq 1$, and $1 \leq q \leq m$. Functions $f_1(x), f_2(x) \in P_k$ are called p^q-*equivalent* iff $f_1(x) - f_2(x) \equiv 0 (\mathrm{mod}\ p^q)$.

The following Theorem 3 is a basis for the construction of our recognition algorithm.

Theorem 3. *Let $k = p^m$, where p is a prime number, $m \geq 1$. Let $f(y, x_1, \ldots, x_n) \in P_k$, $f(j, x_1, \ldots, x_n) \in Pol_k$ for each $j \in \mathbb{Z}_k$. Let X be a maximal monomial of non-zero summands in the canonical polynomials of all functions $f(j, x_1, \ldots, x_n)$, it occurs with the coefficient a^j in the canonical polynomial of the function $f(j, x_1, \ldots, x_n)$, and $h(y)$ is such function that $h(j) \equiv a^j (\mathrm{mod}\ k)$, $j \in \mathbb{Z}_k$.*

Then the function $f(y, x_1, \ldots, x_n) \in Pol_k$ iff

1) for the function $h(y)$ there exists $p^{m-cd(X)}$-equivalent for the function $t(y) \in Pol_k$;

2) $f(y, x_1, \ldots, x_n) - X \cdot t(y) \in Pol_k$.

Proof. 1. The necessity. Let $f(y, x_1, \ldots, x_n) \in Pol_k$. Consider its canonical polynomial (by Theorem 2) $F(y, x_1, \ldots, x_n)$ and group summands in it so as to obtain an expression in the form

$$F(y, x_1, \ldots, x_n) = \sum_{i=1}^{l} T_i(y) \cdot X_i,$$

where $T_1(y), \ldots, T_n(y) \in \mathbb{Z}_k[y]$, and $X_1, \ldots, X_l$ are monomials in variables $x_1, \ldots, x_n$. Let X_l be a maximal monomial among $X_1, \ldots, X_l$ in this expression, and $X_l = X_\sigma$ for a certain $\sigma \in \mathbb{Z}_+^n$.

For each $j \in \mathbb{Z}_k$ the canonical polynomial of the function $f(j, x_1, \ldots, x_n)$ can be obtained from the polynomial

$$F(j, x_1, \ldots, x_n) = \sum_{i=1}^{l} T_i(j) \cdot X_i$$

by operations described in the proof of Theorem 2 [9]. Namely, for the summand X_l from its coefficient $p^{m-cd(X_l)}$ is subtracted as long as its coefficient becomes less than $p^{m-cd(X_l)}$.

Now we prove that the monomial X_l has at least one non-zero $(\mathrm{mod}\ k)$ coefficient in the canonical polynomials of the functions $f(j, x_1, \ldots, x_n)$, $j \in \mathbb{Z}_k$, i.e. we must show that $t_l(j) \not\equiv 0 (\mathrm{mod}\ p^{m-cd(X_l)})$ for some $j \in \mathbb{Z}_k$.

Consider the polynomial $T_l(y) = b_s y^s + \ldots + b_1 y + b_0 \in \mathbb{Z}_k[y]$, where $b_s, \ldots, b_1, b_0 \in \mathbb{Z}_k$ are coefficients, $b_s \not\equiv 0 (\mathrm{mod}\ k)$, $s \geq 0$. Since the polynomial F of the function f is canonical, we conclude $b_i < p^{m-cd(y^i \cdot X_l)}$ for all $i = s, \ldots, 1, 0$. So, by virtue of the fact that $cd(y^i X_l) \geq cd(y^i)$, we obtain $b_i < p^{m-cd(y^i)}$.

Therefore, the polynomial $T_l(y)$ is canonical. Consequently, it defines the function that is not the zero (mod k).

In addition, if we suppose that $t_l(y) \equiv 0(\text{mod } p^{m-cd(X_l)})$, we obtain $t_l(y) \cdot g_\sigma(x_1, \ldots, x_n) \equiv 0(\text{mod } p^m)$. The monomial $y^s \cdot X_l$ is a maximal monomial of non-zero summands in the polynomial $T_l(y) \cdot G_\sigma(x_1, \ldots, x_n)$ and occurs in it with the coefficient that is less than $p^{m-cd(y^s X_l)}$. This contradicts to Lemma 4.

Hence, $t_l(y) \not\equiv 0(\text{mod } p^{m-cd(X_l)})$, q.e.d.

From which the monomial X_l is a maximal monomial among non-zero summands in the canonical polynomials of the functions $f(j, x_1, \ldots, x_n)$, $j \in \mathbb{Z}_k$. Therefore, without the loss of generality, $X = X_l$.

Thus, $t(y) = t_l(y)$ is required so $t_l(y) \in Pol_k$ and

$$t_l(y) - h(y) \equiv 0(\text{mod } p^{m-cd(X_l)}).$$

Then

$$f(y, x_1, \ldots, x_n) - X \cdot t_l(y) \in Pol_k$$

as the difference of two polynomial functions.

2. The sufficiency. Suppose there exists a function $t(y) \in Pol_k$ such as $t(y) - h(y) \equiv 0(\text{mod } p^{m-cd(X_l)})$.

Then

$$f(y, x_1, \ldots, x_n) = (f(y, x_1, \ldots, x_n) - X \cdot t(y)) + X \cdot t(y) \in Pol_k$$

as the sum of two polynomial functions.

Remark 1. Theorem 3 suggests the idea of our algorithm.

Let $k = p^m$, where p is a prime number, $m \geq 1$. Let $f(y, x_1, \ldots, x_n) \in P_k$ and $f(j, x_1, \ldots, x_n) \in Pol_k$ for each $j \in \mathbb{Z}_k$. Let $F_j(x_1, \ldots, x_l) = \sum_{i=1}^{l} a_i^j X_i$ be the canonical polynomial for the function $f(j, x_1, \ldots, x_n)$, $j \in \mathbb{Z}_k$.

Then let X_l be a maximal monomial among $X_1, \ldots, X_l$, and $X_l = X_\sigma$ for a certain $\sigma \in \mathbb{Z}_+^n$.

Assume that $h_i(y)$, $i = 1, \ldots, l$, are such functions that $h_i(j) \equiv a_i^j(\text{mod } k)$, $j \in \mathbb{Z}_k$. Note that the functions $h_i(y)$ are not necessarily polynomial.

From which

$$f(y, x_1, \ldots, x_n) \equiv \sum_{i=1}^{l} h_i(y) \cdot X_i(\text{mod } k).$$

Assume $h_l = h$.

Let us consider the sufficiency in Theorem 3. We have $t(y) - h(y) \equiv 0(\text{mod } p^{m-cd(X_l)})$.

Consider the polynomial $G_\sigma(x_1, \ldots, x_n) = X_l + G_1(x_1, \ldots, x_n) \in \mathbb{Z}_k[x_1, \ldots, x_n]$, where all summands in polynomial G_1 are less than X_l in the partial order of monomials in $x_1, \ldots, x_n$.

Then $g(y, x_1, \ldots, x_n) = (t(y) - h(y)) \cdot g_\sigma(x_1, \ldots, x_n) \equiv 0(\text{mod } p^m)$.

Hence, for the function $f(y, x_1, \ldots, x_n) - t(y) \cdot X_l$ we can observe

$$f(y,x_1,\dots,x_n)-t(y)\cdot X_l \equiv (f(y,x_1,\dots,x_n)-t(y)\cdot X_l)+(t(y)-h(y))\cdot g_\sigma(x_1,\dots,x_n) \equiv$$

$$\equiv \sum_{i=1}^{l} h_i(y)\cdot X_i - t(y)\cdot X_l + (t(y)-h(y))\cdot (X_l + g_1(x_1,\dots,x_n)) \equiv$$

$$\equiv \sum_{i=1}^{l-1} h_i(y)\cdot X_i + (t(y)-h(y))\cdot g_1(x_1,\dots,x_n) (\mathrm{mod}\ k). \qquad (1)$$

It follows that the canonical polynomials of the functions $f(j,x_1,\dots,x_n) - t(j)\cdot X_l$, $j \in \mathbb{Z}_k$, have no the monomial X_l or monomials which are greater than X_l. So, we can recursively apply to $f(y,x_1,\dots,x_n) - t(y)\cdot X_l$ the same Theorem 3. □

Now we study *circuit complexity* of solving PRP, i.e. we consider circuits with gates in some basis of P_k as a model of computation. A *complexity* $T(S)$ of a circuit S is a number of gates in it.

Theorem 4. *Let k be fixed, and $k = p^m$, where p is a prime number, $m \geq 2$.*

For a fixed basis of P_k for every natural number n it can construct a circuit S_n with k^n inputs $u(\sigma)$, $\sigma \in \mathbb{Z}_k^n$, and $k^n + 1$ outputs $v(\sigma)$, $\sigma \in \mathbb{Z}_k^n$, and z, such that if values $f(\sigma)$ of an arbitrary function $f \in P_k^n$ appear on inputs $u(\sigma)$ then z gives 1 in the case $f \in Pol_k$ and z gives 0 otherwise, and, furthermore, if $z = 1$ then outputs $v(\sigma)$ give coefficients of monomials X_σ in the canonical polynomial of a function f, and the family of circuits $S_1, S_2, \dots, S_n, \dots$ has the complexity $T(S_n) = O(k^n)$.

Proof. We describe an algorithm by which it can construct such circuit S_n.

In the description of the sign "':='" means "'assignment"', i.e. in the variable to the left of the sign the expression value to the right of the sign is written.

In addition, we apply the following operations: $\min(x,y)$ is finding the minimum of the two elements in $\mathbb{Z}_k$; $\mathrm{mod}(x,y)$ is taking the remainder of the left operand modulo its right operand (operands is in $\mathbb{Z}_k$). Note that these operations can be computed in some circuits with constant complexity.

For $j \in \mathbb{Z}_k$, $\sigma = (s_1,\dots,s_n) \in \mathbb{Z}_k^n$ we assume $(j,\sigma) = (j,s_1,\dots,s_n) \in \mathbb{Z}_k^{n+1}$.

For $f(y,x_1,\dots,x_n) \in P_k^{n+1}$ we denote a function $f(j,x_1,\dots,x_n)$, $j \in \mathbb{Z}_k$, as f_j and call it a *subfunction* of a function f.

In the description of the algorithm we assume that if $f \notin Pol_k$, then the values of outputs $v(\sigma)$ are arbitrary.

We describe the algorithm by induction on n. Note that a circuit S_1 can be constructed with constant complexity.

Denote by $SPE(q)$, $1 \leq q < m$, a circuit with k inputs $u(j)$, $j \in \mathbb{Z}_k$, and $k+1$ outputs $v(j)$, $j \in \mathbb{Z}_k$, and z, such that if values $h(j)$ of an arbitrary function $h \in P_k^1$ appear on inputs $u(j)$ then z gives 1 if there exists the p^q-equivalent to h a function $t \in Pol_k^1$ and z gives 0 otherwise, and, furthermore, if $z = 1$ then outputs $v(j)$ give coefficients of monomials $X_{(j)}$ in the canonical polynomial of such function t. Note that for every $q = 1,\dots,m-1$, a circuit $SPE(q)$ can be constructed with constant complexity.

We describe the inductive remove from n to $n+1$ step by step. Let $u(\tau)$, $v(\tau)$, z, where $\tau \in \mathbb{Z}_k^{n+1}$, be inputs and outputs of a circuit S_{n+1} respectively. Below we show by operator "':='" how values of outputs $v(\tau)$ and z are changed to get their result values.

Step 1. Checking the existence of polynomial representations for subfunctions.

Take k circuits S_n by inductive hypothesis and denote them S_n^j, $j \in \mathbb{Z}_k$. Then let $u^j(\sigma)$, $v^j(\sigma)$, z^j be their inputs and outputs respectively, $j \in \mathbb{Z}_k$.

Assume that $u(j,\sigma) = u^j(\sigma)$. Then $z := \min\limits_{j \in \mathbb{Z}_k} z^j$ and $v(j,\sigma) := v^j(\sigma)$ for all $j \in \mathbb{Z}_k$, $\sigma = (s_1, \ldots, s_n) \in \mathbb{Z}_k^n$.

Step 2. Construction polynomials for functions in one variable (by Theorem 3 and Remark 1).

For all $\sigma = (s_1, \ldots, s_n) \in \mathbb{Z}_{mp}^n$ for all monomials X_σ, in descending in some linear order of monomials containing their partial order, from the largest one to the smallest one, repeat the following operations.

Let X_σ be a considered monomial.

Let $h(y)$ be such function that $h(j) \equiv v(j,\sigma) (\mathrm{mod}\ k)$, $j \in \mathbb{Z}_k$. Bring on inputs of a circuit $SPE(m - cd(X_\sigma))$ values of the function h, and let $w(j)$, $j \in \mathbb{Z}_k$, z_σ be its outputs.

Then $z := \min(z, z_\sigma)$ and $v(j,\sigma) := w(j)$.

Let $G_\sigma(x_1, \ldots, x_n) = X_\sigma + \sum\limits_{i=1}^{l} b_i X_{\sigma_i}$, where $\sigma_i < \sigma$. Recall that $g_\sigma \equiv 0 (\mathrm{mod}\ p^{cd(X_\sigma)})$, and by Lemma 2 $\mathrm{len}(G_\sigma) \le (mp)^m$.

Then $v(j,\sigma_i) := v(j,\sigma_i) + (t(j) - h(j)) \cdot b_i$ where $j \in \mathbb{Z}_k$, $i = 1, \ldots, l$ (see Remark 1, the equation (1)).

Step 3. Reduction of a polynomial to the canonical form (by Theorem 2).

For all $j \in \mathbb{Z}_k$ for all $\sigma = (s_1, \ldots, s_n) \in \mathbb{Z}_{mp}^n$ for all monomials X_τ, where $\tau = (j,\sigma) \in \mathbb{Z}_k \times \mathbb{Z}_{mp}^n$, descending in some linear order of monomials containing their partial order, from the largest one to the smallest one, repeat the following operations.

Let X_τ be a considered monomial, and $G_\tau(y, x_1, \ldots, x_n) = X_\tau + \sum\limits_{i=1}^{l} d_i X_{\tau_i}$, where $\tau_i < \tau$.

1) If $cd(X_\tau) \ge m$, and $cd(X_\sigma) < m$, then $g_\tau \equiv 0 (\mathrm{mod}\ p^m)$. By Lemma 2 $\mathrm{len}(G_\tau) \le p^m (mp)^m$.

Then $v(\tau_i) := v(\tau_i) - v(\tau) \cdot d_i$ for $i = 1, \ldots, l$, and after that $v(\tau) := 0$

2) If $cd(X_\tau) < m$, then by Lemma 2 $\mathrm{len}(G_\tau) \le (mp)^m$.

Then $v(\tau_i) := v(\tau_i) - (v(\tau) - (v(\tau) \bmod p^{m - cd(X_\tau)})) \cdot d_i$, for $i = 1, \ldots, l$, and after that $v(\tau) := v(\tau) \bmod p^{m - cd(X_\tau)}$.

The description of the algorithm is finished. Now $z = 1$ if $f \in Pol_k$, and $z = 0$ otherwise. Furthermore, if $z = 1$ then outputs $v(\tau)$ give coefficients of monomials X_τ, $\tau \in \mathbb{Z}_k^{n+1}$.

The correctness of this algorithm follows from Theorem 3 and Remark 1.

Now we evaluate the complexity of the circuit S_{n+1}.

Let S_n^1, S_n^2, S_n^3 be the circuits, respectively implementing steps 1, 2 and 3 of the algorithm. Then for their complexities $T(S_n^i)$, $i = 1, 2, 3$, we obtain:

$$T(S_n^1) \leq kT(S_n) + C_1,$$

where C_1 is a constant (as S_n^1 consists of k circuits S_n and computes the value of variable z),

$$T(S_n^2) \leq C_2(mp)^n,$$

where C_2 is a constant (as S_n^2 takes no more than $(mp)^n$ monomials X_σ to perform a constant number of operations with every monomial)

$$T(S_n^3) \leq C_3 k(mp)^n,$$

where C_3 is a constant (as S_n^3 takes no more than $k(mp)^n$ monomials X_τ to perform a constant number of operations with every monomial).

Then

$$\begin{gathered} T(S_{n+1}) \leq T(S_n^1) + T(S_n^2) + T(S_n^3) \leq kT(S_n) + C_4(mp)^n \leq \\ \leq k(kT(S_{n-1}) + C_4(mp)^{n-1}) + C_4(mp)^n \leq \ldots \leq \\ \leq k^n T(S_1) + C_4(k^{n-1}(mp) + \ldots + (mp)^n) \leq \\ \leq Ck^n \left(\sum_{i=1}^{n} \left(\frac{mp}{p^m} \right)^i \right), \end{gathered}$$

where C_4, C are constants.

Then, if $k > 4$, then $mp < p^m$. Hence we can limit the expression in the large brackets by the sum of the corresponding infinite decreasing geometric progression.

Therefore, $T(S_n) = O(k^n)$ with $k > 4$.

In the case where $mp = p^m$, that is when $k = 4$, we carry out further investigation. Note that

$$(x^2 + x)(y^2 + y) \equiv 0 (\mathrm{mod}\ 2^2).$$

Therefore, if a monomial X consists of at least two variables in powers greater than unity, then $cd(X) \geq 2$ (with respect to $p = 2$). Therefore, in steps 2 and 3 we can consider exclusively such $\sigma = (s_1, \ldots, s_n) \in \mathbb{Z}_4^n$, in which no more than one coordinate is more than 1. There exists $2^n + 4n2^{n-1} \leq C_5 n2^n$, such monomials, where C_5 is a constant.

Then

$$\begin{gathered} T(S_{n+1}) \leq T(S_n^1) + T(S_n^2) + T(S_n^3) \leq 4T(S_n) + C_6 n2^n \leq \\ \leq 4(4T(S_{n-1}) + C_6(n-1)2^{n-1}) + C_6 n2^n \leq \ldots \leq \\ \leq 4^n T(S_1) + C_6(4^{n-1}2 + \ldots + n2^n) \leq \\ \leq C4^n \left(\sum_{i=0}^{n} i \left(\frac{1}{2} \right)^i \right), \end{gathered}$$

where C_6, C are constants.

We can limit the expression in the large brackets by the sum of the corresponding infinite convergent series.

Therefore, $T(S_n) = O(k^n)$ with $k = 4$.

Hence, $T(S_n) = O(k^n)$, if $k = p^m$, where p is a prime number, $m \geq 2$.

Remark 2. Let $k = p_1^{m_1} \cdot \ldots \cdot p_q^{m_q}$, where p_i are different prime numbers, $m_i \geq 1$, $i = 1, \ldots, q$, $q \geq 2$.

Then, since the ring $\mathbb{Z}_k$ is the direct sum of ideals which is isomorphic to the rings $\mathbb{Z}_{p^{m_i}}$, applying the algorithm of Theorem 4 we can construct a similar algorithm with linear complexity (i.e. the family of circuits S_n with complexity $T(S_n) = O(k^n)$) to check the existence of polynomial representation for functions in P_k, and, if so, to find their canonical polynomials modulo k. □

Our algorithm is especially elegant if $k = 4$, since, in this case, in Theorem 3 the condition of the existence a function $t(y) \in Pol_k$, which is $p^{m-cd(X)}$-equvivalent to a function $h(y)$, is replaced on the condition $h(y) \in Pol_k$ by the fact $2\chi_i(y) \in Pol_4$ for every $i \in \mathbb{Z}_4$, where $\chi_i(i) \equiv 1 (\mathrm{mod}\ 4)$ and $\chi_i(x) \equiv 0 (\mathrm{mod}\ 4)$ for $x \not\equiv i (\mathrm{mod}\ 4)$.

Example 1. Consider the function $f(x, y) \in P_4$ and apply the algorithm of Theorem 4 to it.

For the convenience we draw the Table in which the column "'xy"' keeps values on $\sigma \in \mathbb{Z}_4^2$, the column "'$f$"' keeps values $f(\sigma)$, the column "'$v1$"' contains coefficients of the canonical polynomials $f(j, y)$, $j \in \mathbb{Z}_4$, of monomials $X_{(j,y)}$, and, finally, the column "'$v2$"' gives coefficients of the canonical polynomial $f(x, y)$ of monomials $X_{(x,y)}$.

The inductive behavior of the algorithm is described below the Table.

xy	f	$v1$	$v2$
00	3	3	3
01	3	0	0
02	3	0	0
03	3	0	0
10	0	0	0
11	3	3	2
12	2	0	0
13	1	0	0

xy	f	$v1$	$v2$
20	3	3	0
21	3	0	1
22	3	0	0
23	3	0	0
30	2	2	1
31	1	3	0
32	0	0	0
33	3	0	0

Then $F(3, y) = 3y + 2$, $F(2, y) = 0$, $F(1, y) = 3y$, $F(0, y) = 3$.

And $V1(x, 3) = V1(x, 2) = 0$, $V1(x, 1) = x^2 + 2x$, $V1(x, 0) = x^3 + 3$.

We get the polynomial $F(x, y) = 3 + 2xy + x^2y + x^3$.

It was already written in the canonical form.

Example 2. Consider another function $g(x, y) \in P_4$ and apply the algorithm of Theorem 4 to it.

xy	g	$v1$	$v2$
00	2	2	
01	0	1	
02	0	0	
03	0	**1**	
10	0	0	
11	0	0	
12	0	0	
13	0	**0**	

xy	g	$v1$	$v2$
20	0	0	
21	0	0	
22	0	0	
23	0	**0**	
30	0	0	
31	0	0	
32	0	0	
33	0	**0**	

Then $G(3, y) = G(2, y) = G(1, y) = 0$, $G(0, y) = y^3 + y + 2$.
But $v1(x, 3) \notin Pol_4$ (in the Table the function $v1(x, 3)$ is indicated in bold.)
Hence $g(x, y) \notin Pol_4$.

Acknowledgements. The author would like to thank her reviewers for helpful criticism that contributed to the preparation of the improved final version of the article.

The author is very grateful to V.A. Zakharov, Dr.Sc., for the information about CSR-2012 and for help in writing the introduction for the first version of the article.

The author is also grateful to M.A. Kruglova, Prof. of English, for the control of the English language in the article.

References

1. Niven, I., Warren, L.J.: A Generalization of Fermat's Theorem. Proceedings of the American Mathematical Society 8(2), 306–313 (1957)
2. Carlitz, L.: Functions and Polynomials (mod p^n). Acta Arithmetica 9, 66–78 (1964)
3. Aizenberg, N.N., Semyon, I.V., Tsitkin, F.I.: Cardinality of the Class of Functions of k-valued Logic in n Variables which Represented by Polynomials Modulo k. In: The Book: High Stability Elements and Their Applications. Sov. Radio, pp. 79–83 (1971) (in Russian); also transl.: Polynomial Representations of Logical Functions. Automatic Control and Computer Sciences 5, 5–11 (1971)
4. Anzenberg, N.N., Semyon, I.V.: Some Criterium for Representation of k-valued Logic Functions by Polynomials Modulo k. In: The Book: High Stability Elements and Their Applications. Sov. Radio, pp. 84–88 (1971) (in Russian)
5. Singmaster, D.: On Polynomial Functions (mod m). Journal of Number Theory 6, 345–352 (1974)
6. Rosenberg, I.G.: Polynomial Functions over Finite Rings. Glasnik Matematiki 10(1), 25–33 (1975)
7. Remizov, A.B.: On a Superstructure of the Closed Class of Polynomials Modulo k. Discrete Mathematics 1(1), 3–15 (1989) (in Russian); in English: Discrete Mathematics and Applications 1(1), 9–22 (1991)
8. Meschaninov, D.G.: A Method of Constructing Polynomials for Functions of k-valued Logic. Discrete Mathematics 7(3), 48–60 (1995) (in Russian); in English: Discrete Mathematics and Applications 5(4), 333–346 (1995)
9. Selezneva, S.N.: Fast Algorithm of Construction Polynomials Modulo k for k-valued Functions if k is a Composite Number. Discrete Mathematics 23(3), 3–22 (2011) (in Russian)

Boolean Composition of Visual Secret Sharing Schemes

Hans Ulrich Simon

Fakultät für Mathematik, Ruhr-Universität Bochum
D-44780 Bochum, Germany
hans.simon@rub.de

Abstract. In this paper, we analyze the disjunction and the conjunction of two access structures for visual secret sharing schemes. The latter operation, when applied to k-out-of-n schemes, leads to schemes with multiple thresholds. We precisely determine the maximum relative contrast for schemes of this type. As in the case of classical schemes with a single threshold, the analysis proceeds by revealing a central relation between the relative contrast in a visual secret sharing scheme and the error-term in a related problem of approximation-theoretic flavor.

1 Introduction

Visual Cryptography and k-out-of-n secret sharing schemes (so-called threshold schemes) are notions introduced by Naor and Shamir [5]. A sender wishing to transmit a secret message distributes n transparencies among n participants of the scheme, where the single transparencies contain seemingly random pictures. A k-out-of-n scheme achieves the following situation. If any k participants stack their transparencies together, then a secret message is revealed visually. On the other hand, if only $k-1$ participants stack their transparencies, or analyze them by any other means, they are not able to obtain any information about the secret message.

The basic model by Naor and Shamir has been extended and generalized in several directions. For example, the concept of "general access structures" [1] allows for a flexible specification of the sets of participants who are qualified for the visual disclosure of the secret. Another variation are the schemes with "perfect reconstruction of black pixels". They are motivated by studies on visual perception and were suggested first in [2].
The main results in this paper are as follows:

1. We consider the conjunction (resp. disjunction) Γ of two given access structures Γ' and Γ'' and show that the corresponding best possible relative contrast values satisfy the relation $\alpha \geq \frac{1}{2}\alpha'\alpha''$ (resp. $\alpha \geq \min\{\alpha', \alpha''\}$).
2. We show that these relations hold with equality for every disjunction of two arbitrary schemes and every conjunction of two threshold schemes.
3. We extend these results to the Boolean composition of more than two access structures.

E. Hirsch et al. (Eds.): CSR 2012, LNCS 7353, pp. 314–325, 2012.

In order to show the main result about conjunction of two threshold schemes, we apply the same key techniques that had been used for the determination of the maximum relative contrast, say $\alpha(k,n)$, achievable in a k-out-of-n scheme: By means of *symmetrization*, it was shown in [3] that $\alpha(k,n)$ can be cast as the maximum gain in a (cleverly designed) linear program. By means of *dualization*, it was shown in [4] that $\alpha(k,n)$ can be cast as the minimum error in a well-known approximation-theoretic problem.
We show that these techniques, when properly used, lead to a tight analysis for threshold schemes with multiple thresholds. The connections to approximation theory that are uncovered on the way may find some interest in their own right.

The intimate relations between visual cryptography and approximation theory provide us with versatile and powerful instruments for the analysis of visual secret sharing schemes. In particular, upper bounds on the maximum relative contrast are found simply by providing feasible solutions for the corresponding (dual) approximation-theoretic problems. A meta-goal in this paper is to bundle the known key techniques and to demonstrate their amazing power.

2 Definitions, Notations and Facts

Let us first fix some general notation. For positive integers n and $n_1 \leq n_2$, $[n]$ denotes the set $\{1,\ldots,n\}$ and, more generally, $[n_1 : n_2]$ denotes the set $\{n_1, n_1+1, \ldots, n_2\}$. The least common multiple of two positive integers a, b is denoted $\mathrm{lcm}(a,b)$. The powerset of a set S is denoted 2^S. We say a set T is *maximal* in a collection $K \subseteq 2^S$ of subsets of S, if T belongs to K but no proper superset of T does. δ denotes the Kronecker symbol, i.e., $\delta(a,b) = 1$ if $a = b$, and $\delta(a,b) = 0$ otherwise. For a matrix A with k rows, $A^\top$ denotes its transpose, $A[i]$ denotes its i-th row, and, for every $I \subseteq [k]$, $A[I]$ denotes the submatrix formed by the rows $A[i]$ such that $i \in I$. The entry of A located in the i-th row and the j-th column is denoted $A[i,j]$. For vectors $u, v \in \quad^m$, $\langle u, v\rangle$ denotes their scalar product, i.e., $\langle u, v\rangle = \sum_{i=1}^m u_i v_i$. For a given decomposition $k = k_1 + k_2$ of the number k of rows, a block-decomposition $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$ implicitly means that A_1 consists of the first k_1 rows of A, and A_2 consists of the last k_2 rows of A. The analogous convention holds for vectors. For Boolean vectors $v, w \in \{0,1\}^m$, $H(w) = \sum_{i=1}^m w_i$ denotes the Hamming-weight of w and $v \vee w = (v_i \vee w_i)_{i\in[m]}$ denotes the entry-wise Boolean disjunction of v and w. A vector with non-negative real coefficients that sum-up to 1 is called a *probability vector*. $\mathcal{P}_k$ denotes the set of univariate polynomials of degree at most k over the reals.

2.1 Visual Secret Sharing Schemes

In this subsection, we recall the definition of general access structures and visual secret sharing schemes from [1].

A *general access structure for a set* $[n]$ *of participants* is pair $\Gamma = (\mathcal{Q}, \mathcal{F})$ such that $\mathcal{Q}, \mathcal{F} \subseteq 2^{[n]}$ and $\mathcal{Q} \cap \mathcal{F} = \emptyset$. $\mathcal{Q}$ contains the so-called *qualified sets*, and $\mathcal{F}$

contains the so-called *forbidden sets*. $\mathcal{F}$ is monotonically decreasing (i.e., every subset of a forbidden set is forbidden too). $\mathcal{F}_0$ denotes the maximal forbidden sets. We consider two examples that will play a major role in this paper:

Example 1. For $\mathcal{Q} = \{X \subseteq [n] : \ |X| = k\}$, a set is qualified iff it consists of k participants. Clearly, $\mathcal{F}_0 = \{X \subseteq [n] : \ |X| = k-1\}$.

Example 2. Let $n = n_1 + n_2$ and $\mathcal{Q} = \{X \subseteq [n] : \ |X \cap [n_1]| = k_1, |X \cap [n_1 + 1 : n_2]| = k_2\}$. Here a set is qualified if it contains k_1 participants from the "first group" (with numbers between 1 and n_1) and k_2 participants from the second (with numbers between $n_1 + 1$ and n_2). Note that $\mathcal{F}_0$ consists of all sets with either $k_1 - 1$ participants from the first group and all participants from the second, or with $k_2 - 1$ participants from the second group and all participants from the first.

Let $\Gamma = (\mathcal{Q}, \mathcal{F})$ be an access structure for n participants. A (Γ, m)-*scheme* is given by two multisets of $(n \times m)$-matrices, $\mathcal{C}_0, \mathcal{C}_1$, a function $t : \mathcal{Q} \to [m]$, and a parameter $0 < \alpha \leq 1$ such that the following holds:

1. For every $X \in \mathcal{Q}$ and all $A \in \mathcal{C}_0, B \in \mathcal{C}_1$, $H(\vee_{i \in X} A[i]) \leq t(X) - \alpha m$ and $H(\vee_{i \in X} B[i]) \geq t(X)$.
2. For every $X \in \mathcal{F}$, the multisets $\mathcal{C}_0[X] = \{A[X] : \ A \in \mathcal{C}_0\}$ and $\mathcal{C}_1[X] = \{B[X] : \ B \in \mathcal{C}_1\}$ contain the same matrices with the same frequencies.

We refer to the first (resp. second) condition as the "visibility condition" (resp. "security condition"). Note that it suffices to check the security condition for all $X \in \mathcal{F}_0$. The parameter m is called the *pixel expansion* of the scheme; α is called the *relative contrast*. If there exist matrices $G_0, G_1 \in \{0,1\}^{n \times m}$ such that, for $b = 0, 1$, $\mathcal{C}_b$ consists of the $m!$ matrices obtained from G_b by a permutation of the columns, we say that the scheme has *generator matrices* G_0, G_1. If the function $t : \mathcal{Q} \to [m]$ assigns value m to every set $X \in \mathcal{Q}$, we say the scheme *perfectly reconstructs black pixels*. A scheme for the access structure from Example 1 is simply called k-*out-of-n* or (k, n)-*scheme* in this paper. A scheme for the access structure from Example 2 is called $(k_1, n_1; k_2, n_2)$-scheme. The maximum relative contrast achievable in a (k, n)-scheme (resp. $(k_1, n_1; k_2, n_2)$-scheme) is denoted $\alpha(k, n)$ (resp. $\alpha(k_1, n_1; k_2, n_2)$).

A (Γ, m)-*scheme* is used as follows to achieve the situation described in the introduction. A (black or white) pixel of a secret image is decomposed into m subpixels which are arranged in a fixed pattern (e.g., a rectangle). If the sender wishes to transmit a white pixel, she randomly chooses a matrix $A \in \mathcal{C}_0$ and transmits $A[i]$ to the i-th participant. The handling of black pixels is analogous: the i-th participant obtains the row $B[i]$ of a matrix B randomly chosen from $\mathcal{C}_1$. The security condition makes sure that, for forbidden sets of participants, the transmission of a white pixel is statistically indistinguishable from the transmission of a black pixel. The visibility condition makes sure that participants of a qualified set may stack their transparencies together so that a black pixel has a fraction of black sub-pixels (actually fraction 1 in case of

perfect reconstruction) that exceeds the fraction of black sub-pixels of a white pixel by at least α. It is obviously desirable to maximize the relative contrast.[1]

2.2 Contrast as Objective Function in a Linear Program

The following result casts the problem of maximizing the relative contrast as an instance of Linear Programming:

Theorem 1 ([3]). *$\alpha(k,n)$ equals the maximal value of*

$$\alpha = \sum_{j=0}^{n-k} \binom{n-k}{j} \binom{n}{j}^{-1} (x_j - y_j) \tag{1}$$

subject to the following constraints:

(1-LP1) *x and y are probability vectors.*
(1-LP2) *For all $l = 0, \ldots, k-1$, $\sum_{j=l}^{n-k+l+1} \binom{n-k+1}{j-l} \binom{n}{j}^{-1} (x_j - y_j) = 0$.*

The essential features of the linear program from Theorem 1 are nicely captured by the following definition of a linear program $\mathrm{LP}_{c,A}(k,n)$:

Maximize $\langle c, x-y \rangle = \sum_{j=0}^{n} c_j (x_j - y_j)$
subject to (1-LP1) and $A(x-y) = 0$.

Here, c, x, y are vectors with components c_j, x_j, y_j, respectively, and with j ranging from 0 to n. A is an arbitrary real-valued matrix with k rows and $1+n$ columns. We say $\mathrm{LP}_{c,A}(k,n)$ is of *type 1-BAP (with polynomials $P \in \mathcal{P}_k$ and $Q_0, \ldots, Q_{k-1} \in \mathcal{P}_{k-1}$ as "witnesses")* if the following conditions are satisfied:

(1-BAP1) For all $j = 0, \ldots, n$, $c_j = P(j)$.
(1-BAP2) For all $l = 0, \ldots, k-1$ and all $j = 0, \ldots, n$, $A[l,j] = Q_l(j)$.
(1-BAP3) Every polynomial of degree at most $k-1$ is a linear combination of the polynomials $Q_0, \ldots, Q_{k-1}$.

We denote the value of an optimal solution for $\mathrm{LP}_{c,A}(k,n)$ by $\alpha_{c,A}(k,n)$ in what follows.

Theorem 2 ([4]). *Maximizing the value of α in (1) subject to the constraints (1-LP1), (1-LP2) is of type 1-BAP.*

Note that $\alpha_{c,A}(k,n)$ collapses to $\alpha(k,n)$ — the maximum relative contrast achievable by a (k,n)-scheme — when c and A are chosen so as to obtain the linear program from Theorem 1.
For every $P \in \mathcal{P}_k$, we define the following function in k, n:

$$\beta_P(k,n) = \min_{Q \in \mathcal{P}_{k-1}} \max_{j=0,\ldots,n} |P(j) - Q(j)| \tag{2}$$

A polynomial in $\mathcal{P}_{k-1}$ that minimizes the right hand-side of (2) is called a *best approximating polynomial in $\mathcal{P}_{k-1}$ for $P \in \mathcal{P}_k$*. The following result links together visual cryptography and approximation theory:

[1] In this paper, we sweep the conflicting goal of minimizing the pixel expansion under the carpet.

Theorem 3 ([4]). *If* $\mathrm{LP}_{c,A}(k,n)$ *is of type 1-BAP and* P *is the polynomial which witnesses (1-BAP1), then* $\alpha_{c,A}(k,n) = 2 \cdot \beta_P(k,n)$.

3 The Composition of Two Access Structures

Let Γ' and Γ'' be two access structures with disjoint sets of participants. The access structure $\Gamma' \wedge \Gamma'' = (\mathcal{Q}, \mathcal{F})$, called the *(Boolean) conjunction of* Γ' *and* Γ'', is given by $\mathcal{Q} = \{X \cup Y : X \in \mathcal{Q}' \wedge Y \in \mathcal{Q}''\}$ and $\mathcal{F} = \{X \cup Y : X \in \mathcal{F}' \vee Y \in \mathcal{F}''\}$. The (Boolean) disjunction (already discussed in [1]) is defined analogously by setting $\mathcal{Q} = \{X \cup Y : X \in \mathcal{Q}' \vee Y \in \mathcal{Q}''\}$ and $\mathcal{F} = \{X \cup Y : X \in \mathcal{F}' \wedge Y \in \mathcal{F}''\}$. Note that the access structure for a $(k_1, n_1; k_2, n_2)$-scheme is the conjunction of the access structure for a (k_1, n_1)-scheme and the access structure for a (k_2, n_2)-scheme. The *r-fold replication of a multiset* $\mathcal{C}$ of matrices is the multiset $\mathcal{C}(r) = \{[C_1|\cdots|C_r] : C_1, \ldots, C_r \in \mathcal{C}\}$ where the multiplicity of the composed matrix is the product of the multiplicities of the r building blocks. The *r-fold replication of a* (Γ, m)*-scheme* given by $\mathcal{C}_0, \mathcal{C}_1$ is the (Γ, rm)-scheme given by $\mathcal{C}_0(r), \mathcal{C}_1(r)$. r-fold replication increases the pixel expansion by factor r but leaves the relative contrast unchanged. The first main result in this paper is as follows:

Theorem 4. *A* (Γ', m')*-scheme achieving relative contrast* α' *and a* (Γ'', m'')*-scheme achieving relative contrast* α'' *can be transformed into a* $(\Gamma' \wedge \Gamma'', 2 \cdot \mathrm{lcm}(m', m'')^2\})$*-scheme (called the* conjunction of the two given schemes*) achieving relative contrast* $\frac{1}{2}\alpha'\alpha''$*. Moreover this transformation preserves perfect reconstruction of black pixels, i.e., if the given two schemes perfectly reconstruct black pixels then the composed scheme does this too.*

Proof. Let $m = \mathrm{lcm}(m', m'')$. Thanks to the replication trick, we easily get a (Γ', m)- and a (Γ'', m)-scheme, say the former is given by $\mathcal{C}'_0, \mathcal{C}'_1$ and the latter by $\mathcal{C}''_0, \mathcal{C}''_1$. (Recall that replication leaves the relative contrast unchanged.) For a matrix M with m columns, $R_0(M), \ldots, R_{m-1}(M)$ denote the m matrices obtained from M by cyclic rotations of the columns. We now define the multisets of matrices, $\mathcal{C}_0$ and $\mathcal{C}_1$, that represent the desired $(\Gamma' \wedge \Gamma'', 2m^2\})$-scheme. Let A' (resp. B', A'', B'') range over all matrices in $\mathcal{C}'_0$ (resp. $\mathcal{C}'_1, \mathcal{C}''_0, \mathcal{C}''_1$). Then, by definition, $\mathcal{C}_0$ contains all matrices of the form

$$A = \left[\begin{array}{c|c|c|c|c|c|c} A' & \cdots & A' & B' & \cdots & B' \\ R_0(A'') & & R_{m-1}(A'') & R_0(B'') & & R_{m-1}(B'') \end{array} \right] \tag{3}$$

and all matrices that are mirror-images of (3) in the sense that the final m blocks in (3) come first and the first m blocks in (3) come last. Similarly, $\mathcal{C}_1$ contains all matrices of the form

$$B = \left[\begin{array}{c|c|c|c|c|c|c} A' & \cdots & A' & B' & \cdots & B' \\ R_0(B'') & & R_{m-1}(B'') & R_0(A'') & & R_{m-1}(A'') \end{array} \right] \tag{4}$$

together with the corresponding mirror images. In both multisets, the multiplicity of the composed matrix is, by definition, the product of the multiplicities of A', B', A'', B'', respectively. The theorem is now obtained from the following

Claim 1: $\mathcal{C}_0, \mathcal{C}_1$ satisfy the security condition and achieve relative contrast $\frac{1}{2}\alpha'\alpha''$. Moreover, if $\mathcal{C}'_0, \mathcal{C}'_1$ and $\mathcal{C}''_0, \mathcal{C}''_1$ perfectly reconstruct black pixels, then $\mathcal{C}_0, \mathcal{C}_1$ does this too.

The proof of the claim is found in Section A of the appendix. □

The transformation in Theorem 4 creates very large multisets $\mathcal{C}_0, \mathcal{C}_1$. Fortunately, these multisets can often be stored succinctly by means of an implicit representation. An example illustrating implicit representations and, of course, the transformation itself is found in Section B of [6].

As proved in Section C of [6], a result analogous to Theorem 4 holds for the disjunction of two access structures. Here, the relative contrast achieved by the composed scheme is $\min\{\alpha', \alpha''\}$.
The following result is an immediate consequence of Theorem 4:

Corollary 1. *Let Γ' (resp. Γ'') be an access structure with maximum relative contrast α'_{max} (resp. α''_{max}). Then, the maximum relative contrast α_{max} of a scheme for $\Gamma' \wedge \Gamma''$ satisfies $\alpha_{max} \geq \frac{1}{2}\alpha'_{max}\alpha''_{max}$. This relation holds correspondingly for schemes that perfectly reconstruct black pixels.*

4 Conjunction of Two Threshold-Schemes

The main result in this section, Theorem 7, implies that the transformation from Theorem 4 is "contrast-optimal" (i.e., transforms two schemes achieving the maximum relative contrast, respectively, into a composed scheme which does this too) whenever it is applied to threshold schemes. The following result is the counterpart to Theorem 1:

Theorem 5. *$\alpha(k_1, n_1; k_2, n_2)$ equals the maximal value of*

$$\alpha = \sum_{j_1=0}^{n_1-k_1} \sum_{j_2=0}^{n_2-k_2} \binom{n_1-k_1}{j_1} \binom{n_2-k_2}{j_2} \binom{n_1}{j_1}^{-1} \binom{n_2}{j_2}^{-1} (x_{j_1,j_2} - y_{j_1,j_2}) \qquad (5)$$

subject to the following constraints:

(2-LP1) *x and y are (doubly indexed) probability vectors.*
(2-LP2) *For all $l_1 = 0, \ldots, k_1 - 1$ and all $l_2 = 0, \ldots, n_2$,*

$$\sum_{j_1=l_1}^{n_1-k_1+l_1+1} \binom{n_1-k_1+1}{j_1-l_1} \binom{n_1}{j_1}^{-1} (x_{j_1,l_2} - y_{j_1,l_2}) = 0 \ .$$

For all $l_1 = 0, \ldots, n_1$ and all $l_2 = 0, \ldots, k_2 - 1$,

$$\sum_{j_2=l_2}^{n_2-k_2+l_2+1} \binom{n_2-k_2+1}{j_2-l_2} \binom{n_2}{j_2}^{-1} (x_{l_1,j_2} - y_{l_1,j_2}) = 0 \ .$$

The proof of this theorem is similar to the proof Theorem 1. It is found in Section D of [6].

The essential features of the linear program from Theorem 5 are nicely captured by the following definition of a linear program $\mathrm{LP}_{c,A}(k_1, n_1; k_2, n_2)$:

Maximize $\langle c, x - y \rangle = \sum_{j_1=0}^{n_1} \sum_{j_2=0}^{n_2} c_{j_1,j_2}(x_{j_1,j_2} - y_{j_1,j_2})$
subject to (2-LP1) and $A(x - y) = 0$.

Here c, x, y are vectors (viewed as one-dimensional) with components c_{j_1,j_2}, x_{j_1,j_2}, y_{j_1,j_2}, respectively, and with j_b ranging from 0 to n_b for $b = 1, 2$. A is an arbitrary real-valued matrix with $(1 + n_1)(1 + n_2)$ columns. The columns of A are addressed by the double indices j_1, j_2 so that the format of the rows of A fits with the format of x and y. We say $\mathrm{LP}_{c,A}(k_1, n_1; k_2, n_2)$ is of *type 2-BAP* if there exist vectors $c^{(1)} \in {}^{1+n_1}$, $c^{(2)} \in {}^{1+n_2}$ and matrices $A'_1 \in {}^{k_1 \times n_1}$, $A'_2 \in {}^{k_2 \times n_2}$ such that following holds:

1. $\mathrm{LP}_{c^{(1)},A'_1}(k_1, n_1)$ is of type 1-BAP, say with witnesses $P_1 \in \mathcal{P}_{k_1}$ and $Q_0, \ldots, Q_{k_1-1} \in \mathcal{P}_{k_1-1}$.
2. $\mathrm{LP}_{c^{(2)},A'_2}(k_2, n_2)$ is of type 1-BAP, say with witnesses $P_2 \in \mathcal{P}_{k_2}$ and $R_0, \ldots, R_{k_2-1} \in \mathcal{P}_{k_2-1}$.
3. For all $j_1 = 0, \ldots, n_1$ and all $j_2 = 0, \ldots, n_2$,
$$c_{j_1,j_2} = c^{(1)}_{j_1} \cdot c^{(2)}_{j_2} = P_1(j_1) \cdot P_2(j_2) \ . \tag{6}$$
4. A can be written in the form $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$ such that the following holds:
   (a) The rows of A_1 are addressed (l_1, l_2) with l_1 ranging from 0 to $k_1 - 1$ and l_2 ranging from 0 to n_2. For every row (l_1, l_2) of A_1,
   $$A_1[(l_1, l_2), (j_1, j_2)] = A'_1[l_1, j_1] \cdot \delta(l_2, j_2) = Q_{l_1}(j_1) \cdot \delta(l_2, j_2) \ . \tag{7}$$
   (b) The rows of A_2 are addressed (l_1, l_2) with l_1 ranging from 0 to n_1 and l_2 ranging from 0 to $k_2 - 1$. For every row (l_1, l_2) of A_2,
   $$A_2[(l_1, l_2), (j_1, j_2)] = A'_2[l_2, j_2] \cdot \delta(l_1, j_1) = R_{l_2}(j_2) \cdot \delta(l_1, j_1) \ . \tag{8}$$

We denote the value of an optimal solution for $\mathrm{LP}_{c,A}(k_1, n_1; k_2, n_2)$ by $\alpha_{c,A}(k_1, n_1; k_2, n_2)$ in what follows. We briefly note that the second equalities in (6), (7), (8), respectively, are redundant as they are implied by the first two conditions. We included them explicitly for ease of later reference.

Lemma 1.

$$\alpha_{c,A}(k_1, n_1; k_2, n_2) \geq \frac{1}{2} \cdot \alpha_{c^{(1)},A'_1}(k_1, n_1) \cdot \alpha_{c^{(2)},A'_2}(k_2, n_2) \tag{9}$$

Proof. For $b = 1, 2$, let $x^{(b)}, y^{(b)}$ be an optimal solution for the linear program $\mathrm{LP}_{c^{(b)},A'_b}(k_b, n_b)$. Thus, $x^{(b)}$ and $y^{(b)}$ are probability vectors and $A'_b(x^{(b)} - y^{(b)}) = 0$. For $j_1 = 0, \ldots, n_1$ and $j_2 = 0, \ldots, n_2$, let

$$x_{j_1,j_2} = \frac{1}{2} \cdot (x^{(1)}_{j_1} x^{(2)}_{j_2} + y^{(1)}_{j_1} y^{(2)}_{j_2}) \ \text{ and } \ y_{j_1,j_2} = \frac{1}{2} \cdot (x^{(1)}_{j_1} y^{(2)}_{j_2} + y^{(1)}_{j_1} x^{(2)}_{j_2}) \ .$$

Note that

$$x_{j_1,j_2} - y_{j_1,j_2} = \frac{1}{2} \cdot (x^{(1)}_{j_1} - y^{(1)}_{j_1}) \cdot (x^{(2)}_{j_2} - y^{(2)}_{j_2}) \ . \tag{10}$$

The theorem is now obtained from the following

Claim 2: x, y is a feasible solution for the linear program $\mathrm{LP}_{c,A}(k_1, n_1; k_2, n_2)$, and the target function α evaluated at x, y yields the right hand-side of (9).

The proof of Claim 2 is found in Section B of the appendix. □

Theorem 6. *Maximizing the value of α in (5) subject to the constraints (2-LP1), (2-LP2) is of type 2-BAP.*

Proof. The problem is obviously of the form $\mathrm{LP}_{c,A}(k_1, n_1; k_2, n_2)$. In particular (compare with (5)),

$$c_{j_1,j_2} = \underbrace{\binom{n_1 - k_1}{j_1}\binom{n_1}{j_1}^{-1}}_{=:c^{(1)}_{j_1}} \cdot \underbrace{\binom{n_2 - k_2}{j_2}\binom{n_2}{j_2}^{-1}}_{=:c^{(2)}_{j_2}} \ . \tag{11}$$

Moreover (compare with (2-LP2)), A can be written in the form $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$ where

$$A_1[(l_1, l_2), (j_1, j_2)] = \underbrace{\binom{n_1 - k_1 + 1}{j_1 - l_1}\binom{n_1}{j_1}^{-1}}_{=:A'_1[l_1,j_1]} \cdot \delta(l_2, j_2) \ , \tag{12}$$

$$A_2[(l_1, l_2), (j_1, j_2)] = \underbrace{\binom{n_2 - k_2 + 1}{j_2 - l_2}\binom{n_2}{j_2}^{-1}}_{=:A'_2[l_2,j_2]} \cdot \delta(l_1, j_1) \ . \tag{13}$$

According to Theorem 2, the linear programs $\mathrm{LP}_{c^{(b)},A'_b}(k_b, n_b)$ are of type 1-BAP for $b = 1, 2$. Our discussion shows that the problem we started with is of type 2-BAP. □

An inspection of the proof of Theorem 6 shows that $\alpha_{c,A}(k_1, n_1; k_2, n_2)$ collapses to $\alpha(k_1, n_1; k_2, n_2)$ when c is chosen according to (11) and $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$ is chosen according to (12), (13).

The next step is dualization. To this end, we consider again an arbitrary linear program of type 2-BAP (witnessed by polynomials $P_1, P_2, Q_0, \ldots, Q_{k_1-1}$, $R_0, \ldots, R_{k_2-1}$). The two equality constraints in (2-LP1) lead to the dual variables t_1 and t_2, respectively. As for the equality constraints $A(x - y) = 0$, we create the dual variable u_{l_1,l_2} for every row $(l_1, l_2) \in [0 : k_1 - 1] \times [0 : n_2]$ of A_1 and a dual variable v_{l_1,l_2} for every row $(l_1, l_2) \in [0 : n_1] \times [0 : k_2 - 1]$ of A_2. Now dualization leads to the following problem:

Minimize $t_1 + t_2$
subject to $A_1^\top u + A_2^\top v + (t_1, \ldots, t_1)^\top \geq c$ and $A_1^\top u + A_2^\top v - (t_2, \ldots, t_2)^\top \leq c$.

We refer to $A_1^\top u + A_2^\top v + (t_1, \ldots, t_1)^\top \geq c$ (resp. $A_1^\top u + A_2^\top v - (t_2, \ldots, t_2)^\top \leq c$) as the "$x$-constraints" (resp. "$y$-constraints") because they correspond to the primal variables x_{j_1,j_2} (resp. y_{j_1,j_2}).

The vector $A_1^\top u + A_2^\top v$ has an approximation-theoretic interpretation which becomes evident from the following calculation for an arbitrary but fixed index pair (j_1, j_2):

$$\begin{aligned}(A_1^\top u)_{j_1,j_2} &= \sum_{l_2=0}^{n_2} \sum_{l_1=0}^{k_1-1} u_{l_1,l_2} A_1[(l_1,l_2),(j_1,j_2)] \\ &\overset{(7)}{=} \sum_{l_2=0}^{n_2} \sum_{l_1=0}^{k_1-1} u_{l_1,l_2} Q_{l_1}(j_1)\delta(l_2,j_2) \\ &= \sum_{l_1=0}^{k_1-1} u_{l_1,j_2} Q_{l_1}(j_1) =: Q^{(j_2)}(j_1)\end{aligned}$$

Since u is a vector of unconstrained variables and $Q_0, \ldots, Q_{k_1-1}$ span the space of all polynomials of degree at most $k_1 - 1$, the outcome of our calculation, $Q^{(j_2)}(j_1)$, is an arbitrary polynomial of degree at most $k_1 - 1$ in variable j_1. It is important to note that there are $n_2 + 1$ such polynomials at our disposal: one for each value of j_2. A similar calculation shows that, for every fixed j_1, $(A_2^\top v)_{j_1,j_2}$ can be viewed as an arbitrary polynomial, say $R^{(j_1)}(j_2)$, of degree at most $k_2 - 1$ in variable j_2. It follows from this discussion and from $c_{j_1,j_2} = P_1(j_1)P_2(j_2)$ that the x_{j_1,j_2}-constraint (with $j_b = 0, \ldots, n_b$ for $b = 1, 2$) can be rewritten as

$$P_1(j_1)P_2(j_2) - (Q^{(j_2)}(j_1) + R^{(j_1)}(j_2)) \leq t_1 \ , \tag{14}$$

and, similarly, the y_{j_1,j_2}-constraint can be rewritten as

$$(Q^{(j_2)}(j_1) + R^{(j_1)}(j_2)) - P_1(j_1)P_2(j_2) \leq t_2 \ . \tag{15}$$

Consider the following approximation problem for given polynomials $P_1 \in \mathcal{P}_{k_1}$ and $P_2 \in \mathcal{P}_{k_2}$: find polynomials $Q^{(0)}, \ldots, Q^{(n_2)} \in \mathcal{P}_{k_1-1}$ and $R^{(0)}, \ldots, R^{(n_1)} \in \mathcal{P}_{k_2-1}$ which minimize

$$\max_{j_1=0,\ldots,n_1; j_2=0,\ldots,n_2} \left| P_1(j_1)P_2(j_2) - (Q^{(j_2)}(j_1) + R^{(j_1)}(j_2)) \right| \ . \tag{16}$$

The smallest possible value of (16) is denoted $\beta_{P_1,P_2}(k_1, n_1; k_2, n_2)$. With this notation, the following holds:

Lemma 2. *Consider a linear program* $\mathrm{LP}_{c,A}(k_1, n_1; k_2, n_2)$ *of type 2-BAP. Let* $P_1 \in \mathcal{P}_{k_1}$ *and* $P_2 \in \mathcal{P}_{k_2}$ *be the polynomials satisfying (6). Then the following holds:*

$$\alpha_{c,A}(k_1, n_1; k_2, n_2) = 2 \cdot \beta_{P_1,P_2}(k_1, n_1; k_2, n_2) \tag{17}$$

Proof. By duality, the optimal value of the primal, $\alpha_{c,A}(k_1, n_1; k_2, n_2)$, equals the optimal value of the dual. It is evident from (14) and (15) that the cost function in the dual, $t_1 + t_2$, measures the worst approximation error occurring on pairs $(j_1, j_2) \in [0 : n_1] \times [0 : n_2]$ when we try to approximate $P_1(j_1)P_2(j_2)$ by a polynomial of the form $Q^{(j_2)}(j_1) + R^{(j_1)}(j_2)$: t_1 (resp. t_2) accounts for errors where the approximation underestimates (resp. overestimates) the true value. The lemma now follows by observing that the constant terms in the polynomials $Q^{(j_2)}, R^{(j_1)}$ can be chosen so as to bring the two error terms, t_1 and t_2, into balance. □

Lemma 3. *For all polynomials $P_1 \in \mathcal{P}_{k_1}$, $P_2 \in \mathcal{P}_{k_2}$, the following holds:*

$$\beta_{P_1,P_2}(k_1, n_1; k_2, n_2) \leq \beta_{P_1}(k_1, n_1) \cdot \beta_{P_2}(k_2, n_2)$$

Proof. For $b = 1, 2$, let $P_b^* \in \mathcal{P}_{k_b-1}$ denote the best approximating polynomial for P_b. For

$$Q^{(j_2)}(j_1) = P_2(j_2)P_1^*(j_1) \quad \text{and} \quad R^{(j_1)}(j_2) = P_2^*(j_2)(P_1(j_1) - P_1^*(j_1)) \ , \tag{18}$$

we get

$$P_1(j_1)P_2(j_2) - (Q^{(j_2)}(j_1) + R^{(j_1)}(j_2)) = (P_1(j_1) - P_1^*(j_1)) \cdot (P_2(j_2) - P_2^*(j_2)) \ .$$

Since the absolute value of the latter expression is bounded by $\beta_{P_1}(k_1, n_1) \cdot \beta_{P_2}(k_2, n_2)$ for all $j_1 = 0, \ldots, n_2$ and all $j_2 = 0, \ldots, n_2$, the lemma follows. □

We are ready now for the main result in this section:

Theorem 7. *The following equalities are valid:*

$$\beta_{P_1,P_2}(k_1, n_1; k_2, n_2) = \beta_{P_1}(k_1, n_1) \cdot \beta_{P_2}(k_2, n_2) \tag{19}$$

$$\alpha_{c,A}(k_1, n_1; k_2, n_2) = \frac{1}{2} \cdot \alpha_{c^{(1)},A_1'}(k_1, n_1) \cdot \alpha_{c^{(2)},A_2'}(k_2, n_2) \tag{20}$$

$$\alpha(k_1, n_1; k_2, n_2) = \frac{1}{2} \cdot \alpha(k_1, n_1) \cdot \alpha(k_2, n_2) \tag{21}$$

Proof. According to Lemma 3, $\beta_{P_1,P_2}(k_1, n_1; k_2, n_2) \leq \beta_{P_1}(k_1, n_1) \cdot \beta_{P_2}(k_2, n_2)$. The converse direction is obtained as follows:

$$\beta_{P_1,P_2}(k_1, n_1; k_2, n_2) \overset{(17)}{=} \frac{1}{2} \cdot \alpha_{c,A}(k_1, n_1; k_2, n_2) \overset{(9)}{\geq}$$

$$\frac{1}{4} \cdot \alpha_{c^{(1)},A_1}(k_1,n_1 \cdot \alpha_{c^{(2)},A_2}(k_2,n_2 \overset{(17)}{=} \beta_{P_1}(k_1, n_1) \cdot \beta_{P_2}(k_2, n_2)$$

Equation (20) follows directly from (19) and (17). According to Theorem 6, (21) is a special case of (20). □

An extension of our results to the Boolean composition of more than two access structures is found in Section F of [6].

We conjecture that the equalities in Theorem 7 hold correspondingly for schemes with perfect reconstruction of black pixels. But, although it easy to adjust the linear program accordingly, the resulting approximation-theoretic problem is surprisingly hard to analyze.

References

1. Ateniese, G., Blundo, C., De Santis, A., Stinson, D.R.: Visual cryptography for general access structures. Information and Computation 129(2), 86–106 (1996)
2. Henk, E.R., van Tilborg, H.C.A.: Constructions and properties of k out of n visual secrete sharing schemes. Design, Codes and Cryptography 11(2), 179–196 (1997)
3. Hofmeister, T., Krause, M., Simon, H.U.: Contrast-optimal k out of n secret sharing schemes in visual cryptography. Theoretical Computer Science 240(2), 471–485 (2000)
4. Krause, M., Simon, H.U.: Determining the optimal contrast for secret sharing schemes in visual cryptography. Combinatorics, Probability and Computing 12(3), 285–299 (2003)
5. Naor, M., Shamir, A.: Visual Cryptography. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 1–12. Springer, Heidelberg (1995)
6. Simon, H.U.: Boolean composition of visual secret sharing schemes, full version (2012), `http://www.ruhr-uni-bochum.de/lmi/simon/publications/others.html`

A Proof of Claim 1

As for the security condition, let us consider a forbidden set $X \cup Y$ such that $X \in \mathcal{F}'$ or $Y \in \mathcal{F}''$. Let us assume that $X \in \mathcal{F}'$ and $Y \in \mathcal{Q}''$. (The other cases are similar.) Fix a matrix $A \in \mathcal{C}_0$ of the form (3). Then, $A[X \cup Y]$ equals

$$\left[\begin{array}{c|c|c|c|c|c|c} A'[X] \\ R_0(A'')[Y] \end{array}\right| \cdots \left|\begin{array}{c} A'[X] \\ R_{m-1}(A'')[Y] \end{array}\right| \begin{array}{c} B'[X] \\ R_0(B'')[Y] \end{array} \left|\cdots\right| \begin{array}{c} B'[X] \\ R_{m-1}(B'')[Y] \end{array}\Bigg] \quad . \tag{22}$$

(The case of the mirror image is similar.) If $A'[X]$ occurs m'_0 times in $\mathcal{C}'_0[X]$, $B'[X]$ occurs m'_1 times in $\mathcal{C}'_1[X]$, $A''[Y]$ occurs m''_0 times $\mathcal{C}''_0[Y]$, and $B''[Y]$ occurs m''_1 times in $\mathcal{C}''_1[Y]$, then, since $X \in \mathcal{F}'$, $A'[X]$ occurs m'_0 times in $\mathcal{C}'_1[X]$ and $B'[X]$ occurs m'_1 times in $\mathcal{C}'_0[X]$. Notice that $A''[Y] \neq B''[Y]$ because $Y \in \mathcal{Q}''$. It follows that $A[X \cup Y]$ occurs $m'_0 m'_1 m''_0 m''_1$ times in $\mathcal{C}_0[X \cup Y]$. Note that, since $A'[X]$ occurs m'_0 times in $\mathcal{C}'_1[X]$, we may alternatively think of $A'[X]$ as a sub-matrix of the form $\tilde{B}'[X]$ where $\tilde{B}' \in \mathcal{C}'_1$. Similarly, we may think of $B'[X]$ as a submatrix of the form $\tilde{A}'[X]$ where $\tilde{A}' \in \mathcal{C}'_0$. Performing these substitutions in (22), we arrive at a sub-matrix that is a mirror image of (4) (with $\tilde{A}'$ in the role of A' and $\tilde{B}'$ in the role of B'). The punchline of this discussion is that (22) occurs $m'_0 m'_1 m''_0 m''_1$ times in $\mathcal{C}_1[X \cup Y]$ (the same frequency that we had analyzed for $\mathcal{C}_0[X \cup Y]$). Thus, the security condition is satisfied.

We move on to the visibility condition. Let us assume that $X \in \mathcal{Q}'$ and $Y \in \mathcal{Q}''$. It suffices to show that, for all matrices A, B of the form (3) resp. (4), the number of zero-columns in $A[X \cup Y]$ exceeds the number of zero-columns in $B[X \cup Y]$ by at least $\alpha' \alpha'' m^2$ (which leads to the desired relative contrast $\frac{1}{2}\alpha' \alpha''$). According to our assumptions, $B'[X]$ has at least $t'(X)$ nonzero-columns whereas the number of nonzero-columns in $A'[X]$ is at most $t'(X) - \alpha' m$. Let $s' = m - t'(X)$ and $s'' = m - t''(Y)$. With this notation, $A'[X]$ has at least $s' + \alpha' m$ zero-columns whereas the number of zero-columns in $B[X]$ is at most s'. The analogous remark applies to $A''[Y]$ and $B''[Y]$. Note that the rotations in (3) make sure that every

zero-column in $A'[X]$ (resp. $B'[X]$) is aligned with every zero-column in $A''[Y]$ (resp. $B''[Y]$) exactly once. Similarly, the rotations in (4) make sure that every zero-column in $A'[X]$ (resp. $B'[X]$) is aligned with every zero-column in $B''[Y]$ (resp. $A''[Y]$) exactly once. It follows that $A[X \cup Y]$ has at least $(s' + \alpha' m) \cdot (s'' + \alpha'' m) + s's''$ zero-columns whereas the number of zero-columns in $B[X \cup Y]$ is at most $(s' + \alpha' m) \cdot s'' + (s'' + \alpha'' m) \cdot s'$. The difference between these values is at least $\alpha' \alpha'' m^2$, as desired. As for the preservation of perfectly reconstructing black pixels, we briefly observe that a scheme perfectly reconstructs black pixels iff, for every qualified set X and every $B \in \mathcal{C}_1$, the submatrix $B[X]$ does not contain any zero-column. Next, we observe that replication and rotation (of columns) preserve the matrix-property of not having any zero-column. We finally observe that the matrices from $\mathcal{C}_1$, like (4) or the corresponding mirror-image, have a (possibly rotated) matrix from either $\mathcal{C}'_1$ or $\mathcal{C}''_1$ in each of the $2m$ blocks. Thus these matrices do not contain any zero-column provided that none of the matrices from $\mathcal{C}_1$ or $\mathcal{C}''_1$ does this.

B Proof of Claim 2

Obviously, x, y satisfy condition (2-LP1). The following calculation, focusing on a fixed row (l_1, l_2) of A_1, verifies the condition $A_1(x - y) = 0$:

$$\begin{aligned}
(A_1(x-y))_{l_1,l_2} &= \sum_{j_2=0}^{n_2} \sum_{j_1=0}^{n_1} A_1[(l_1,l_2),(j_1,j_2)](x_{j_1,j_2} - y_{j_1,j_2}) \\
&\overset{(7),(10)}{=} \frac{1}{2} \cdot (x^{(2)}_{l_2} - y^{(2)}_{l_2}) \cdot \sum_{j_1=0}^{n_1} Q_{l_1}(j_1) \cdot (x^{(1)}_{j_1} - y^{(1)}_{j_1}) \\
&\overset{(1-BAP2)}{=} \frac{1}{2} \cdot (x^{(2)}_{l_2} - y^{(2)}_{l_2}) \cdot (A'_1(x^{(1)} - y^{(1)}))_{l_1} = 0
\end{aligned}$$

Condition $A_2(x-y) = 0$ can be verified analogously. The proof is now completed by the following calculation:

$$\begin{aligned}
2 \cdot \langle c, x - y \rangle &= 2 \cdot \left(\sum_{j_1=0}^{n_1} \sum_{j_2=0}^{n_2} c_{j_1,j_2}(x_{j_1,j_2} - y_{j_1,j_2}) \right) \\
&\overset{(6),(10)}{=} \prod_{b=1,2} \left(\sum_{j_b=0}^{n_b} P_b(j_b) \cdot (x^{(b)}_{j_b} - y^{(b)}_{j_b}) \right) \\
&\overset{(1-BAP1)}{=} \left\langle c^{(1)}, x^{(1)} - y^{(1)} \right\rangle \cdot \left\langle c^{(2)}, x^{(2)} - y^{(2)} \right\rangle
\end{aligned}$$

Since $x^{(b)}, y^{(b)}$ was chosen as an optimal solution for $\mathrm{LP}_{c^{(b)},A'_b}(k_b, n_b)$, the latter expression coincides with $\alpha_{c^{(1)},A'_1}(k_1, n_1) \cdot \alpha_{c^{(2)},A'_2}(k_2, n_2)$.

Author Index